

Applicability of Neural Networks to Software Security

A thesis submitted in partial fulfilment
of the requirements for the degree of:

Master of Philosophy

By

Adebiyi A B

0415051

Supervisors

Dr Sin Wee Lee

Dr Haris Mouratidis

Dr Chris Imafidon

UNIVERSITY OF EAST LONDON

SCHOOL OF Architecture, COMPUTING
AND ENGINEERING

Acknowledgement

First and foremost, I would like to express my profound gratitude to God Almighty for making it possible for me get this far. To Him alone be all the glory and honour forever. I also appreciate all the efforts of Director of studies Dr Sin Wee Lee and co-supervisors, Dr Harris Mouratidis and Dr Chris Imafidon. I am indeed very grateful for all your contributions to this research work. I also thank Juliette Lewars and Johnnes Arreymbi for all their support rendered during this research work.

My appreciation goes to Prof Leslie Smith of the University of Stirling for the constructive critique given on this work and his valuable advice during the course of this research work. I also thank Bryan Parno of Microsoft for reviewing this work and also suggesting what to do to further progress this research. My gratitude also goes to Prof David Al-dabass of the University of Cambridge for accepting this research work to be presented in the 14th International Conference on Computer Modelling and Simulation in Cambridge. I also thank Charlotte Hall for her keen interest in this research and for all her effort in sending part of this research work to reviewers who have given their critique. I am indeed very grateful.

I will also like to thank Comrade Shittu Amitolu, consultant to the Osun stae governor, for his keen interest in the implementation of this research work in Osun state, Nigeria and for his invitation to present a proposal based on this research work to the delegates of the Osun state government during their visit to UEL. I consider this opportunity to contribute to the development of ICT in Osun state a great privilege which I very much appreciate.

I thank all my brothers and sisters and my parents Caroline and Peter Adebisi, for all their support during the course of this research work, I am eternally grateful for all what you have done. Lastly to my wife Funke and children Erimi and Asepe, thank you for holding on and staying by me. I could not have made it this far without all the sacrifice you have made for me. Thank you, God bless you.

Adebisi Adetunji

Abstract

Software design flaws account for 50% software security vulnerability today. As attacks on vulnerable software continue to increase, the demand for secure software is also increasing thereby putting software developers under more pressure. This is especially true for those developers whose primary aim is to produce their software quickly under tight deadlines in order to release it into the market early. While there are many tools focusing on implementation problems during software development lifecycle (SDLC), this does not provide a complete solution in resolving software security problems. Therefore designing software with security in mind will go a long way in developing secure software. However, most of the current approaches used for evaluating software designs require the involvement of security experts because many software developers often lack the required expertise in making their software secure.

In this research the current approaches used in integrating security at the design level is discussed and a new method of evaluating software design using neural network as evaluation tool is presented. With the aid of the proposed neural network tool, this research found out that software design scenarios can be matched to attack patterns that identify the security flaws in the design scenarios. Also, with the proposed neural network tool this research found out that the identified attack patterns can be matched to security patterns that can provide mitigation to the threat in the attack pattern.

Table of Contents

Acknowledgement.....	1
Abstract	2
Table of Contents	3
List of Tables	4
List of Figures	6
Publications.....	8
Chapter 1. Introduction to Thesis	9
Chapter 2. Literature Review	22
Chapter 3. Software Design Evaluation by Neural Network	50
Chapter 4. Implementation of Neural Network I	70
Chapter 5. Implementation of Neural Network II	91
Chapter 6. Result and Discussion	113
Chapter 7. Conclusion.....	128
References	132
Appendix.....	140
Appendix I	140
Appendix II	141
Appendix III	144
Appendix IV	147
Appendix V	148
Appendix VI	151
Appendix VII	157

List of Tables

Table 2. 1 Quality of Performance of TSP Projects.....	35
Table 3. 1: List of Security patterns	59
Table 3. 2: Security Patterns matched with Attack Patterns	60
Table 4. 1: List of Attack Attributes	75
Table 4. 2: List of Attack Components	78
Table 4. 3: Sample of Pre-processed Training Data from Attack Scenario	81
Table 4. 4: Sample of Training data after encoding	81
Table 4. 5: Sample of data input in Neural Network.....	82
Table 4. 6: Training and Test data sets	85
Table 4. 7: MSE of Neural Network I with SCG Applied.....	86
Table 4. 8: MSE of Neural Network I with RP Applied	87
Table 4. 9: Number of Epoch used in Neural Network I with SCG Applied	88
Table 4. 10: Number of Epoch used in Neural Network I with RP Applied.....	89
Table 5. 1: Classification of Attack Pattern	92
Table 5. 2: Security Design Pattern by Steel, et.al (2005).....	93
Table 5. 3: Security Design Patterns by Blakley, et.al (2004).....	93
Table 5. 4: Security Design Patterns by Kienzle and Elder (2003)	94
Table 5. 5: Attributes of Regularly Expressed Attack Patterns	95
Table 5. 6: Classification of Security Design Pattern by Blakley, et.al (2004)	95
Table 5. 7: Classification of Security Design Pattern by Kienzle and Elder (2003)	96
Table 5. 8: Classification of Security Design Pattern by Steel, et.al (2005).....	97
Table 5. 9: Sample of Pre-processed training data from attack pattern	98
Table 5. 10: Sample of training data after encoding	98
Table 5. 11: Sample of input data into neural network.....	98
Table 5. 12: Security Design Patterns Group 1.....	99
Table 5. 13: Security Design Patterns Group 2	100
Table 5. 14: Security Design Patterns Group 3.....	100
Table 5. 15: Security Design Patterns Group 4	101
Table 5. 16: Security Design Patterns Group 5.....	102
Table 5. 17: Security Design Patterns Group 6.....	103
Table 5. 18: MSE of Neural Network II with SCG Applied.....	106
Table 5. 19: MSE of Neural Network I with RP Applied	107
Table 5. 20: Number of Epoch used in Neural Network II with SCG Applied	108
Table 5. 21: Number of Epoch used in Neural Network II with RP Applied.....	109
Table 5. 22: Training Time for Neural Network II with SCG Applied	110
Table 5. 23: Training Time for Neural Network II with RP Applied	111

Table 6. 1:Hypothesis proposed for comparing performance of neural network I	113
Table 6. 2:Average of MSE Results of neural network implemented with SCG and RP	114
Table 6. 3:ANOVA Table for Average MSE Result for Neural Network I.....	114
Table 6. 4:Hypothesis proposed for comparing performance of neural network II	114
Table 6. 5:Average of MSE Results of Neural Network II implemented with SCG and RP	115
Table 6. 6:ANOVA Table for Average MSE Result for Neural Network II.....	115
Table 6. 7: Actual and expected output of Neural Network I	115
Table 6. 8:Actual and expected output of Neural Network I with input from design scenario.....	118
Table 6. 9: Actual and expected output of Neural Network II	119
Table 6. 10: Attack attributes for scenario 1	122
Table 6. 11:Attributes of identified attack pattern in scenario 1.....	123
Table 6. 12:Attributes for scenario 2	123
Table 6. 13:Attributes of identified attack pattern in scenario 2.....	124
Table 6. 14:Attributes for scenario 3	125

List of Figures

Figure 1. 1: Real Cost of Software Security	19
Figure 2. 1: Microsoft Threat Modelling	26
Figure 2. 2: Sample of System Design.....	29
Figure 2. 3: TSP-Secure Methodology for Defect Removal (Davis, 2005).....	34
Figure 2. 4: Structure of a Neuron.....	44
Figure 2. 5: Feedforward Network	46
Figure 2. 6: Feedback Network	46
Figure 3. 1: Model Overview	51
Figure 3. 2: Information on CVE Details database showing attack on webmail	53
Figure 3. 3: Information on Security Focus database showing attack on webmail.....	53
Figure 3. 4: Information on Security Tracker database showing attack on webmail.	54
Figure 3. 5: Neural Network 1 Evaluation process steps	58
Figure 4. 1: Pie chart of data collected	72
Figure 4. 2: Analysis of Attack on Webmail using secure troopos approach.....	73
Figure 4. 3: Sequence diagram on Webmail	74
Figure 4. 4: Concept map showing interaction between the groups in which the attack components has been categorized.....	78
Figure 4. 5: The Neural Network Architecture	83
Figure 4. 6: Plot of MSE for Neural Network I during training.....	86
Figure 4. 7: MSE of Neural Network I with SCG Applied.....	87
Figure 4. 8: MSE of Neural Network I with RP Applied.....	88
Figure 4. 9: Number of Epoch used in Neural Network I with SCG Applied	89
Figure 4. 10: Number of Epoch used in Neural Network I with RP applied.....	90
Figure 5. 1: Security Design Patterns Group 1	99
Figure 5. 2: Security Design Patterns Group 2	100
Figure 5. 3: Security Design Patterns Group 3	101
Figure 5. 4: Security Design Patterns Group 4	102
Figure 5. 5: Security Design Patterns Group 5	103
Figure 5. 6: Security Design Patterns Group 6	104
Figure 5. 7: Plot of MSE for Neural Network II.....	106
Figure 5. 8: Number of Epoch used in Neural Network I with SCG Applied	107
Figure 5. 9: MSE of Neural Network II with RP Applied.....	108
Figure 5. 10: Number of Epoch used in Neural Network I with SCG Applied	109
Figure 5. 11: Number of Epoch used in Neural Network II with RP Applied.....	110
Figure 5. 12: Training Time for Neural Network II with SCG Applied	110
Figure 5. 13: Training Time for Neural Network II with RP Applied	111

Figure 6. 1: Actual vs. Expected output of Network 1 of NN I	117
Figure 6. 2: Actual vs. Expected output of Network 2 of NN I	117
Figure 6. 3: Actual vs. Expected output result of design Scenarios evaluated by NNI	119
Figure 6. 4: Actual vs. Expected output of NN II	120
Figure 6. 5: Class diagram of online shopping portal	121
Figure 6. 6: Sequence diagram for product selection	122
Figure 6. 7: Sequence diagram for shopping cart submission	124
Figure 6. 8: Sequence diagram for customer login	125

Publications

1. Adebiyi, A., et.al. (2011), 'Applicability of Neural Network to Software Security', Abstract accepted for the ACT 2012 conference, London, United Kingdom
2. Adebiyi, A., et.al., (2012), 'Evaluation of Software Design using Neural Network', In the proceedings of the 8th International Conference on Web Information Systems and Technologies (WEBIST), Porto, Portugal
3. Adebiyi, A., et.al., (2012), 'Matching Attack pattern to Security Pattern using Neural Network', In the proceedings of the European Conference on Information Warfare and Security (ECIW-2012), Paris, France
4. Adebiyi A., et.al., (2012), 'Applicability of Neural Network to Software Security' In proceedings of the UKSim 14th International Conference on Computer Modelling and Simulation, Cambridge, United Kingdom, pp19-24
5. Adebiyi, A. et.al (2012), Using Neural Network for Security Analysis in Software Design, , In the proceedings of the WORLDCOMP'12 conference, Las Vegas, USA.
6. Adebiyi, A., et.al (2012), 'Security assessment of software design using neural network' International Journal of Advanced Research in Artificial Intelligence (IJARAI), Vol.1.4, pp1-6
7. Adebiyi, A., et.al (2012), 'Using neural network to propose solutions to threats in attack pattern' International Journal of Soft Computing and Software Engineering (JSCSE) Vol.3.1, pp1-11
8. Adebiyi, A., et.al (2012), 'Security evaluation of online shopping cart using neural network' International Journal of Computer Application (IJCA), Vol.6.2, pp83-93
9. Adebiyi, A., et.al (2013), 'A neural network based security tool for analysing software design' Paper submitted and accepted for the 4th Doctoral Conference of Computing, Electrical and Industrial Systems (DoCEIS) 2013.
10. Adebiyi, A., et.al (2013), 'Resolving threats in attack patterns by security patterns using neural network. Abstract accepted for the International Conference on Information and Intelligent Systems (ICIIS) 2013.

Chapter 1. Introduction to Thesis

1.1 Introduction

As software applications continue to expand into all areas of our business and private lives, they have become an essential part of our day to day lives. It is now common to find software applications running our airplanes, communication and transportation systems, bank transactions, business supply chains, medical equipment, house appliances and enterprise management systems. Therefore, it is very important that software function properly in the production field today given the dependence of our society on technology.

In essence, confidentiality, availability, reliability, safety and integrity are essential properties that must be at the core of software applications today. Many companies investing heavily on software applications now take into serious consideration these software properties in order to ensure that they are not investing into software riddled with security vulnerabilities that will pose a great risk to their business. As a result, the quality of software applications is becoming more and more important in the software industry as software consumers demand for reliable software that will continue to function correctly and meet the security demands of today.

This is no surprise because the rate at which software vulnerabilities are discovered and exploited currently is quite alarming. Several software vulnerabilities are being published each week and many business organisations are paying dearly for poor quality software. The Computer Security Institute (CSI) 2007 security survey report revealed that the cost of cybercrime has doubled in the past year. It confirmed that companies reported average annual losses of \$350,424 up sharply from the \$168, 000 reported in the previous year (Richardson, 2007).

An earlier report by The National Institute of Standards and Technology (NIST) in 2002 revealed that poor quality software cost the US economy \$59.5 billion per year (Cusumano, 2004). In the last decade, the damage reported due to Code Red virus in 2001 was estimated to be at \$2.62 billion and that of Melissa virus in 1999 to be \$1.1 billion. The year 2000 Love bug was also estimated to be at \$8.75 billion (Erbshloe, 2002). Also, the cost of software security breaches in the US for 2011 was estimated to be \$48 billion (Jaspreet, 2012). The financial loss due to Sony security breach was estimated to be \$171 million for new protection, legal cost, fines, and customer support programmes (James, 2011). Notable among Microsoft financial losses due a security flaws is the \$200 million loss during its campaign for .Net because of the discovery of a security hole in Visual C++ .Net (Telang and Wattal, 2004).

Apart from the financial losses reported by companies due to software vulnerabilities, they are also negatively affected when the software vulnerabilities are publicly disclosed. This in turn erodes their customers' confidence, create an in-ability to attract and retain customers and further cripple their reputation as they may face class-action lawsuits. To aggravate this issue, David Rice an instructor at the SANS Institute in an interview with Forbes.com proposed that a vulnerability tax should be created on software based on the number and severity of its security flaws in order to force software industries to mend its buggy ways (Greenberg, 2008).

Software security flaws have been attributed to defects unintentionally introduced during SDLC especially during the design and the implementation phase. Therefore, it is now an on-going challenge for the software industries to look into ways through which software defects can be reduced during SDLC in order to produce more secured software. In view of this, Martyn Thomas, Professor of Software Engineering at Oxford University commented that "the only way to reduce costs and to keep projects within plans is to dramatically reduce the error rate at every stage in the development." By doing this he said that "the product is not only cheaper, but higher quality: more secure, more reliable, and easier to maintain" (Croxford, 2005).

1.2. Motivation

Security vulnerabilities have been discovered in from time to time in various software applications after they are deployed. Consequently software security holes have become common and this problem is growing. Malicious attackers have been taken advantage of these vulnerabilities to break into critical systems of cooperate bodies causing havoc with great consequences. In dealing with this problem the underlying factors causing software vulnerabilities making software insecurity need to be addressed. It has been argued that security flaws pose the most concern among the factors making software insecure. According to Noopur Davis, the analysis of Software Engineering Institute (SEI) on programs produced by thousands of developers reveals that even experienced developers inject numerous defects during SDLC. In line with this view, Frank Piessens stated that "analysis of causes of actual incidents shows that many software vulnerabilities can be traced back to a relatively small number of causes: software developers are making the same mistakes over and over again" (Piessens, 2002). These mistakes constitute the security flaws which cause software insecurity.

In the requirement phase of the SDLC for example, Frank stated that software flaws can be introduced because of the absence of risk analysis where software are developed without any security issues in mind or when there is a biased risk analysis (i.e. when risk analysis is carried out by only one stakeholder of a given information system) or due to the presence of unanticipated risks. Davis further affirmed that design and architectural flaws such as inadequate authentication, invalid authorization, incorrect use of cryptography, failure to

protect data, and failure to carefully partition applications are causes of software insecurity. In a broader view, Alan Paller the director of research at the SANS Institute in his comment on the causes of the vulnerabilities stated that it is a result of poor coding, testing and sloppy software engineering (Davis, 2005). Thus, at every phase of SDLC, security flaws can be introduced into software products.

To therefore ensure that software is built securely, Software security has been suggested as a way for building more secured software by integrating security into every phase of software development Lifecycle (SDLC). This approach views security as an emergent property of the software and much effort is dedicated into weaving security into the software all through SDLC (McGraw, 2003)

Reportedly, 50% of security problems in software products today have been found to be design flaws (McGraw, 2006). Design-level vulnerability has been described as the hardest category of software defect to contend with. Moreover, it requires great expertise to ascertain whether or not a software application has design-level flaws which makes it difficult to find and automate (Hoglund and McGraw, 2004). To further buttress this fact, Paul (2011) applying Pareto principle to software security, states that 80% of software defects arises from the 20% of software design flaws. Therefore by finding and fixing the flaws found during the design stage Paul argued that this will considerably mitigate the threat on the software being developed.

In line with this argument many authors have argued that it is much better to find and fix flaws during the early phase of software development because it is more costly to fix the problem at a late stage of software development and much more costly when the software has been deployed (Spampinato et.al, 2008, Mockel and Abdallah, 2011, Gegick and Williams, 2007). To ensure that security is integrated during the design phase of SDLC, many techniques such as architectural risk analysis, threat modelling, attack trees, attack patterns, use of security tools and other approaches have been proposed (see chapter 2 for further discussion).

However, due to limitations of these techniques, the main motivation for investigating the applicability of neural network to software security is highlighted below in the aims and objective of this research work.

- **Aim**

This research aims at investigating how neural networks can be applied as a tool to evaluate Software designs with regards to its security and also propose possible solutions to the identified flaws in the software design. This will enable software developers to have a

feedback on the security of their software design before the implementation phase of the software development lifecycle.

- **Objectives**

- To identify current techniques used for integrating security into software design and their limitations
- To design and train a neural network to match attack patterns to software design scenarios as a means of identifying security flaws in the design scenarios.
- To propose possible solutions to the security flaws identified in design scenarios by training a second neural network to match possible security patterns that can mitigate the threat in the attack pattern matched to the software design scenario.
- To analyse the performance of the trained neural networks by observing their mean square error (MSE), number of epochs and training time.
- To find out the optimal performance of the neural networks by conducting statistical analysis on the performance of the networks when different training optimization are applied to the networks.
- To carry out a validation study to investigate the ability of the neural networks in matching attack patterns to software design scenarios and in matching security design patterns to attack patterns.
- To compare the proposed neural network approach with current approaches used in integrating security software design during SDLC and also carry a case study to demonstrate how the proposed approach can aid software developers to in integrating security into software design.

Method

This thesis documents the creation and design of a neural network that can be used as a tool for the evaluating software design for security flaws and also suggest possible mitigation. Previous researches have shown various ways in which neural network has been used in the area of security. Neural network based applications has been used successfully in the area of network security such as intrusion detection systems (IDS), misuse detection systems and firewalls (Ahmad et. al, 2006, Bivens et.al, 2002). Also in the field of application security, neural network has been proposed to be used as virus detection system and authentication system (Cannady, 1998, Joseph et.al, 2009). The success of neural network in its usage in these applications and its ability makes it a good candidate for predicting security flaws from software design.

With this in mind, this research adopts the creation and design approach which is based on the following five process steps

- **Awareness of Problem:** This step of the creation and design process deals with the recognition of the problem which may come from multiple sources. This could be from previous researches which have identified areas of further research or from studying allied

discipline where there is an opportunity for new findings that would lead to generation of knowledge. In this research, the problem that has been identified by previous research as mentioned earlier is that 50% of security flaws have been identified to be as a result of software design flaws. To address this problem, this research investigates further into two research work carried out by Michael Gegick and Laurie Williams (2006) and Wiesauer and Sametinger (2009). The outcome of this process step is a proposal for a new research endeavour (Vaishnavi and Kuechler, 2007)

- **Suggestion:** During this process step, suggestions are offered on how the problem identified in the first step is to be addressed based on current knowledge or theories in the domain area where the problem has been identified. This step has been described as a creative leap from curiosity in which tentative ideas offering new functionality are envisioned to provide solution (Oates, 2006). The suggestion this research is offering to address the problem of software design flaws is finding how Neural Networks can be used to evaluate software design in order to help software designers reduce the flaws in their software designs to a minimum before they are implemented. This will in turn reduce the security flaws in the software products. The knowledge generated during this step is the tentative design of the solution.
- **Development:** The tentative design proposed in the previous process step is implemented during this process step. The way the tentative design will be implemented depends on the nature of the artefact to be developed. In order to design the Neural Network suggested as a solution to the problem identified in this research, various issues will be considered. Some of these issues includes deciding the best architecture of the Neural Network (e.g. Feed-forward network or Feed-back network), how the neural network is to be trained to recognise flaws in software design and how the data used for the training data is going to be collected and processed. The development of the artefact in this step is the contribution to knowledge.
- **Evaluation:** The artefact developed either fully or partially is evaluated during this process step based on its expected functionality specified in the suggestion (i.e. the second process step) to examine its performance and observe any deviation from expectations. In this research work, the experimental approach will be used to evaluate the developed Neural Network (This is discussed further below). The outcome of the experiment during this stage results into new knowledge.
- **Conclusion:** At this final step of the design process, the result of findings during the process steps are written up and the knowledge gained are identified (Oates, 2006). The knowledge gained according to Vaisgnavi and Kuechler, 2007 can be categorized as “firm” meaning “facts that have been learned and can be repeatedly applied or behaviour that can be

repeatedly invoked” or as “loose ends” that is “anomalous behaviour that defies explanation and may well serve as the subject of further research.”

For training the neural network to evaluate the security of software design, data was collected from various online vulnerability databases (see chapter 3 for further discussion) and this formed the primary method of data collection. Secondary data was collected from various research works discussed in chapter four and five and from the review of literature on software security and neural networks. Information from the literature review provides an evaluation on what has been done in previous research and also open us questions which needs to be addressed.

1.3. Common Security Flaws in Software Design

Many software vulnerabilities have been linked to flaws in the software design. Preventing these flaws from being introduced during the design phase of SDLC will help software developers make their software more secure and save them from making mistakes that will be very costly to correct if the flaws are detected at a later stage of the SDLC. Some of the flaws attributed to software design include:

1.3.1 Weak Access Control

Access control is the way in which an application grants access to its content and functions to different users (Hu, Ferraiolo, Kuhn, 2006). Granting and revoking privileges is a typical way of providing access control. Privileges are described as what allows specific users to access the application to do only what they are allowed to do (Connolly and Begg, 2005) When authorization of users of a software application is not done properly, this could lead to various security breaches. This design flaw allows users or systems to perform actions that they should not perform. The presence of security flaw is not difficult to discover and exploit. All it would take the attacker is to request for access to functions or content which normally he does not have any privilege to access. If he is granted access, he would have discovered a flaw in the access control that can be exploited and the consequence can be disastrous (CWE, 2013). In this case, the attacker would have access to unauthorized content that is not properly protected which he may be able to change or delete, execute arbitrary code or manipulate the application especially if he is granted an administrator (Open Web Application Security Project (OWASP), 2010).

1.3.2 Weak Authentication

Most applications use log-on passwords to authenticate users. However, a flaw in the authentication routine could be exploited by attackers to impersonate legitimate users (OWASP 2010). This flaw could be in the form of exposed account passwords or session IDs. The attack on Gawker’s database system in 2010 is an example of an attack that exploits this vulnerability as the firm had no password policies for her internal users (Chickowski, 2010).

1.3.3 Failure to Validate Input

The lack of input validation in a software system also jeopardizes its security. This is a security weakness in which an application allows foreign inputs which subverts the legitimate use of a subsystem (Pomraning, 2005). Due to this, a malicious attacker taking the advantage of an un-enforced and unchecked assumption an application makes about its inputs could inject malicious code into the application. The purpose of the injected code typically is to bypass or modify the originally intended functionality of the application. This attack becomes more disastrous if the functionality bypassed is the security of the application. Attacks exploiting this vulnerability could crash or confuse the software application and it could also enable attackers to gain access to sensitive information or manipulate the database maliciously. Web-based applications are noted mostly to be liable to these attacks as they need to collect data from users. The flaw occurs when user supplied data are accepted without proper validation. This is abused by attackers who supply malicious data that could contain code, arbitrary query strings or commands to be processed further by the application, which assumes the input is valid (OWASP, 2010) (Jones, 2010).

For different applications, this flaw is exploited by different attacks. Notable among them is the SQL injection which is particularly widespread and dangerous. It is a technique used to exploit web sites that construct SQL statements from user-supplier inputs to query backend database systems. In this attack, an unsuspecting web application blindly accepts malicious database queries which are then forwarded to the database to be executed with the privileges of the application (Shulman, 2006). In this way an attacker can gain unrestricted access to an entire database and can therefore corrupt or destroy the content of the database. This flaw is also exploited by the cross-site scripting (XSS) attack which is also known as the HTML Injection. This occurs when an attacker send a malicious script in form of a script through web applications to different end user. By running malicious script, the attacker can access any cookie, session token or any other critical information on the end user's browser.

1.3.4 Weak Encryption

Software applications are also open to attacks when the encryption algorithms used for protecting their data are not strong enough. This weakness could be exploited by attackers using brute force to access the data. The OWASP report on the top ten application security risks of 2010 states that 'many web applications do not properly protect sensitive data, such as credit cards, SSNs, and authentication credentials, with appropriate encryption or hashing. Attackers may steal or modify such weakly protected data to conduct identity theft, credit card fraud, or other crimes' (OWASP 2010). Another factor contributing to the cause of this flaw is that sometimes the encryption keys are not generated and stored securely. Attackers find the keys easily and compromise all the data that has been protected. According to Open Web Application Security Project (OWASP) a few areas in which developers make mistake leading to insecure storage includes the following

- Failure to encrypt critical data
- Insecure storage of keys, certificates, and passwords
- Improper storage of secrets in memory
- Poor sources of randomness

- Poor choice of algorithm
- Attempting to invent a new encryption algorithm
- Failure to include support for encryption key changes and other required maintenance procedures

1.4. Challenges in Integrating Security into Software Design

There are various challenges facing the integration of security into software design. Some of these challenges underpin the reason why neural network has been proposed as tool for integrating security into software design. These challenges are discussed below.

1.4.1 Need for Security Experts

There is need for security expert to be involved during SDLC for all the current approaches used to develop secure software application. For instance, conducting architectural risk analysis requires the involvement of security experts who will help in identifying the threats to the software technology, review the software for any security issues, investigate how easy it is to compromise the software's security, analyse the impact on assets and business goals should the security of the software be compromised and recommend mitigating measures to either eliminate the risk identified or reduce it to a minimum (McGraw 2006).

As a result, the existing gap between security professionals and software developers is a challenge that must be addressed in order to integrate security into software during SDLC. The disconnection between this two has led to software development efforts lacking critical understanding of current technical security risks (Pemmaraju and McGraw, 2000). And as the environment in which software are deployed becomes more hostile, ignoring security during SDLC means releasing software with many security defects that could have been avoided.

One of the reasons for this gap is because the goals for the two groups are different. The developers, trained to think of functions and features, focus on the functionality of their product and on-schedule delivery. Moreover, security is often thought of as a feature and not as an emergent system property and developers who intend to integrate security into their products often lack the requisite knowledge required in doing so (McGraw 2002). This critical knowledge is possessed by security professionals who have over the years observed system intrusions, dealt with malicious attackers and have studied software vulnerabilities in minute detail because this has been their focus. However, because few security experts are software developers themselves, their security solutions tend to be limited to reactive techniques such as installing software patches and maintaining firewalls (Pemmaraju and McGraw, 2000).

Recognising that developers lack the knowledge and training necessary to assess and improve the reliability, safety and security of software products, some of the current approaches

includes training for developers on security issues. Even then, security experts outside the development team still need to be involved during the software development especially when very sensitive software systems is being developed or when new technologies is being used. External security experts will also help in finding any assumptions that has being made about the target system that may pose security risks (Wiseman, 2006). As the skills of both software developers and security experts both complement each other in building more secure software application it is therefore very important to find avenues for interdisciplinary cooperation between the two groups (Kenneth and McGraw, 2005).

1.4.2 Process issues

To integrate security into the software design most of the current approaches uses a high level architectural design of the target system in order to allow the developers to view the overall component of the system and know how they are connected and how they work. However, this conflict with some software development processes like extreme programming (XP) that sees no need for spending time up front thinking through the architectural design of the target system before the production coding begins. This is because the architecture of the system is taken to evolve spontaneously as the code base evolves (Stephens, 2002) and it is also claimed that the code is the design (McGraw, 2006).

Any software development process similar to XP that spend little or no time on developing a comprehensible high level architectural design of the target system is therefore likely to come up with software design flaws which may not be discovered until the software is deployed. The unit tests conducted during the SDLC may catch the code-level bug but not the wrongness of the design (Stephens, 2002). This problem is exacerbated when the development process migrate to the often repeating 'code-test-debug' phase which could potentially introduce lots of bugs as dependent code is broken when changes are made to the code during each iteration and this could in turn lead to severe cost and timescale overruns (Croxford, 2005).

The high level architectural design of the target system is very important if security is to be integrated into the software design. This is because it encompasses everything about the target system and also documents both the functional and non-functional design decision (Stephens, 2002). At the implementation phase, the big picture of the whole system which the high level architectural design provides help the developers to know how the components fit together. Thus when the unavoidable changes are to be made to the software during development, the high level architectural design will serve as a roadmap that will help the developers to trace out the overall impact of such changes so that flaws in the software design can be avoided.

1.4.3 Tight budget and time to market constraints

Tight budget and time to market are other issues that also pose challenge into integrating security into software. Many times software developers find themselves working under pressure in fast internet time in order to meet the deadline for releasing their software product to the market. Integrating security into software during SDLC during such intense pressure in today's competitive software market is often seen as too lofty or unattainable and thus a waste of time and money by development managers. It is also observed that the management of some software projects have warmed up to software development processes like XP who see the lack of up-front design as a way of saving money (Stephens, 2002). Therefore software developers are forced to live within their development manager's schedule, feature priority and resource constraints (Pemmaraju, Lord and McGraw, 2000).

However, from previous research it is argued that maintenance and evolution costs account up to 90% of software cost (Koskinen, 2003). Also, from a total cost of ownership (TCO) point of view, by spending more during SDLC to build more secured software reduces the cost of maintenance and this subsequently lowers the TCO. In this view Gary McGraw the CTO Cigital Inc stated that "managers who choose to focus all of their attention on minimizing only the development part of the TCO (often to the detriment of the maintenance part) have a tendency to create poor software faster (resulting in an exploding TCO); while those managers who understand the TCO equation properly can let the development expenses rise a little even as the TCO moves down." (McGraw, 2008)

It is also important to note that the maintenance cost is separate from the cost of risks such as litigation, reputation, brand damage and other risks involved in producing insecure software. As a result, the development management can indeed estimate the future cost of maintenance and resolving security flaws after the software has been released and trade this off with early investment in integrating security into the software during SDLC (McGraw, 2008). A good reason for this is because resolving software security problems after delivery has been observed to be 100 times more expensive than finding and resolving them during the requirement and design phase of SDLC as shown in the chart below (Boehm and Basili, 2001). This further contributes into reducing the TCO of the software.

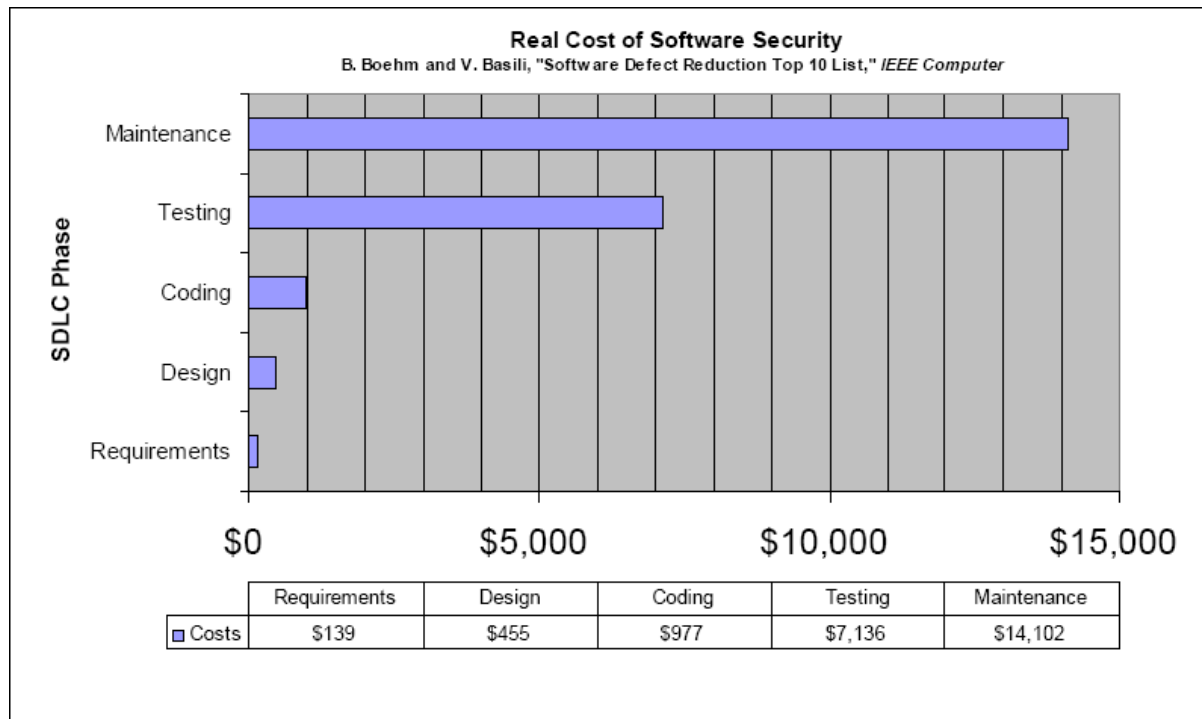


Figure 1. 1: Real Cost of Software Security (Berg 2007)

With regards to time to market also, it has been observed that software projects which aim at producing near defect free software consistently meet their schedule, thereby avoiding the cost involved with delayed releases (Davis, 2005). Contrary to the views that sees integrating security into software during SDLC as slowing down time to market, building secure software given the right level of expertise can sometimes be designed more rapidly than other software system with little or no security at all (McGraw and Viega, 2001).

1.4.4 Tools and technologies for software design analysis

Unlike implementation tools, design tools and technologies for automated analysis of software security at the architectural level has been slow in coming. This is still an area where many researches are currently being undertaken. Because design-level defects are the hardest category of defects to deal with, finding them as been particularly hard to automate (Hoglund and McGraw, 2004). Most of the tools and technologies which are currently available to support software developers in analysing software design for flaws are not widely in use. For these reasons finding software flaws during SDLC still require great human expertise.

1.4.5 Massively distributed system

It is not uncommon today to find complex software applications running on thousands of fat clients connected simultaneously to banks of central servers in the client-server architecture. The growth and complexity of software systems today alone already pose security concerns

because as software systems become larger, bugs cannot be avoided (McGraw, 2002). This problem is exacerbated as these complex software systems become massively distributed, with servers and thousands of users interacting all at the same time. This pushes the limit of software technology especially with regards to state and time thereby posing a serious challenge to software security today.

Security risks arise when the state and time of complex software systems in a distributed environment become entangled with complex trust models as they share their state among distributed processors with different level of trustworthiness (Hoglund and McGraw, 2007). For instance, in transaction based systems which are commonly used in the e-commerce set up; the functionality state of the system is distributed among many components running on several servers. Therefore, with the continuous use of massively distributed systems trust models get more complex. Synchronizing and tracking state will also continue to be an on-going challenge.

The move to web services such as software oriented architecture (SOA), web 2.0 and web 3.0 all the more increase the potential security risks involved in controlling states over trust boundaries. As complex software systems are constructed from mashing up data and functionality all over the web, defining trust boundaries during software design becomes a more challenging endeavour as it becomes more difficult to tell which data can be trusted and which piece of functionality will actually do what they say they do (McGraw, 2008). The massive data mash-up provided by web services on the web poses security concern as they become more exposed because malicious attackers can take the advantage to falsify data, provide services that don't do what they claim and also infiltrate legitimate services. Thus, issues such as revoking identities, privacy and figuring how to evolve trust over time remain open challenges (McGraw, 2008).

Consequently, all these call for serious consideration by software developers during software design. As client software can misbehave or can be manipulated by malicious users, implicit trust assumptions has become a serious security risk which must be dealt with at the design phase. Trust model boundaries for modern software architectures when confused can also generate security problems especially when there is a misunderstanding of what should trust what in the models (Hoglund and McGraw, 2007), (Miller, 2008). For instance, trust model boundaries can become confused when defining trust zones in software systems exhibiting the n-tier architecture relying on several third party components and programming languages in a distributed environment (McGraw, 2006). Unfortunately, these are the software technologies running our banking applications, e-commerce systems, online games and other critical systems today. And with the move to web services, integrating security into the software design for these technologies has become even more challenging.

1.5. Thesis Overview

Chapter two covers discussion on previous researches and technologies that are related to this research work from authoritative sources. To set the background for this research work, much effort has been dedicated into discussing current approaches used in securing software. The approaches include the use of security best practices, security oriented software development methodologies, network Security, application Security, formal methods and security tools. Discussion on neural networks and its application to areas of security that is related to this research work is also presented. Chapter three discusses the proposed neural network approach. The model overview for this research work is presented and each and module of the model overview is discussed. Chapter four and five demonstrates the implementation of the proposed neural networks. Demonstration on the data collection, data encoding, the neural network architecture and training are presented. Chapter six presents the result, analysis and discussion on the performance of the neural network. A statistical analysis on the performance of the neural network is also conducted. Chapter seven provides a summary to this research work and also highlights its benefits and limitations along with suggestion on future work.

1.6. Chapter Summary

This chapter has provided a general back background to this research work. It was noted that most security flaws are caused by software design defects and the need for integrating security in the early phase of SDLC based on the fact that it is cheaper to find and fix the security flaws at this time has been highlighted. The common security flaws in software design includes: weak access control, weak authentication, failure to validate input and weak encryption. Some of the challenges for integrating security into software design were also presented. The creation and design methodology as has been discussed as the method that will be adopted for the research work. The process steps in this approach have been presented. In the next chapter, a literature review will be conducted on the current approaches for integrating security into software design.

Chapter 2. Literature Review

2.1. Introduction

Software insecurity is a big concern to the software industry and to most consumers (Rice, 2007). To resolve the problem, many techniques have been suggested and implemented for integrating security into software applications during software development and after the software has been deployed. In this chapter, a literature review is conducted on some of the techniques used to secure software application during the design phase of the SDLC. Some of the commonly used techniques for providing security after the software has been deployed are also reviewed.

2.2. Integrating security into software design

Some of the security vulnerabilities caused by architectural and design defects includes incorrect use of cryptography, failure to protect data, inadequate authentication, failure to carefully partition applications and invalid authorization (See chapter 1 for further discussion). Most of these defects have been attributed to oversight leading to defect types such as declaration errors, logic errors, loop control errors, conditional expressions errors, failure to validate input, interface specification errors, configuration errors, and failure to understand basic security issues (Davis, 2005). In order to integrate security into software design, different approaches are currently being used in the software industry. Some these approaches are discussed below.

2.2.1 Architectural Risk Analysis

Architectural risk analysis is used to identify vulnerabilities and threats to a software system at the design phase of SDLC, which may be malicious or non-malicious in nature. It examines the preconditions that must be present for the vulnerabilities to be exploited by various threats and assesses the states that the system may enter after a successful attack on the system. One of the advantages of architectural risk analysis is that it enables developers to analyse a software system component by component, tier by tier and environment level by environment level in order to apply the principles of measuring threats, vulnerabilities and impacts at each level (McGraw, 2004). This functional decomposition of the system allows for a desktop review of potential vulnerabilities and also enables the developers to design the high-level architectural view of the system. Higher-level architectural design is also called the forest-level view and it allows the developers to see the big picture of the system thereby enabling them to know how the components are connected and how all the moving parts work (McGraw, 2006). During architectural risk analysis, the high-level design view is used to consider the following important factors:

a) The Assets

These are the resources that need to be protected and these could be in form of data, system components or even a complete system (McGraw, 2004). Traditionally, security practitioners

have been concerned about the confidentiality, integrity, availability and auditability of the assets; however, they also vary in how they are critical in various business organizations (Hope, Lavenhar and Peterson, 2008). For instance, while confidentiality of the data may be the top priority to one organization, availability and integrity of data may be the priority of another organization. For risk analysis to produce good results, the assets that need protection are identified to ensure proper security measures are put in place to protect them.

b) Threats

These are agents or actors who are the source of danger to the security of the system when they carry out attacks on the system. The attacks could be in the form of denial of service, SQL injection or any other form of attack that will eventually compromise the security of the targeted system. Some of the well-known threats include crackers, criminals, malicious hackers and disgruntled employees who violate the protection of assets of a system for a variety of motivations, such as financial gain, prestige, or other motives. Other threats, which are not conscious entities, such as hardware failures, natural disasters, performance delays and user errors are also considered during risk analysis. While all the threat categories are considered, malicious and accidental human activities are mostly considered.

c) Vulnerability

“A vulnerability is a defect or weakness in system security procedures, design, implementation, or internal controls that can be exercised and result in a security breach or violation of security policy” (McGraw, 2006). Vulnerability can exist in one or more components that make up a system and they come in two basic forms, which are:

- **Bugs:** These are implementation-level problems, such as buffer overflow, leading to security risks. Bugs result from the failure to implement the software architecture correctly and these are resolved by fixing the broken lines of code (Hope, Lavenhar and Peterson, 2008). Automated source code analysis tools are used mainly in this area to help in removing bugs in the source code.
- **Flaws:** These are deep-seated failures in the software design that lead to a security risk no matter how well the software is implemented. As stated earlier, architectural flaws are the hardest category of software vulnerability to understand and contend with. As a result, human expertise is required to uncover the flaws.

d) Risk

This is normally calculated as a product of the probability of a threat exploiting a vulnerability and the impact on the organization (i.e. $\text{risk} = \text{probability} \times \text{impact}$). Some of the factors used to determine this calculation include: the ease of executing an attack, the motivation and

resources of an attacker, the existence of vulnerabilities in a system, and the cost or impact in a particular business context (McGraw, 2006).

e) Impact

These are the consequences a business organization must face if there is a successful attack on its software technology by a threat exploiting any vulnerability in the system. This could be primarily expressed in monetary terms as loss of revenue or in terms of negative effects on the business organization marketing abilities in the form of damage to reputation, loss of customer confidence, inability to attract and retain customers, loss of market share and delay or failure in delivery of services as promised within the service-level agreement (SLA). Secondly, the effects of a failing software technology can include increased maintenance cost, increased customer support costs, longer time to market, impact on legal, regulatory and compliance matters and higher cost of development (Hope, Lavenhar and Peterson, 2008).

f) Mitigation

“Risk mitigation refers to the process of prioritizing, implementing, and maintaining the appropriate risk-reducing measures recommended from the risk analysis process.” (Hope, Lavenhar and Peterson, 2008) Mitigating a risk in a software technology means changing the software architecture in one way or another in order to make the system attack-resistant thereby reducing the likelihood or the impact of the risk. Mitigation consists of management, operational and technical controls prescribed for the software technology with the purpose of protecting the system’s software architecture, availability, integrity and confidentiality. These controls may set up either to prevent the risk or to detect the risk when it triggers.

McGraw (2006) presents the architectural risk analysis in three critical steps.

- **Attack resistance analysis:** This involves the use of known attack information in the form of a checklist to identify risks in the architecture during risk analysis.
- **Ambiguity analysis:** This involves the activity needed to discover new risks.
- **Weakness analysis:** This aims at uncovering weaknesses originating from the use of external software platforms.

However, it should be noted that conducting architectural risk analysis requires the involvement of security experts because software developers often lack the knowledge required to integrate security into SDLC.

There are various risk analysis methodologies for software and these are classified into two different groups: commercial and standards-based (McGraw, 2006). Standards-based risk analysis methodologies include:

- ASSET (Automated Security Self-Evaluation Tool) from the National Institute on Standard and Technology (NIST)
- OCTAVE (Operationally Critical Threat, Asset and Vulnerability Evaluation) from the Software Engineering Institute (SEI)
- COBIT (Control Objectives for Information and Related Technology) from the Information System Audit and Control Association (ISACA).

Commercial risk analysis methodologies include:

- STRIDE from Microsoft
- Security Risk Management Guide from Microsoft
- ACSM/SAR (Adaptive Countermeasure Selection Mechanism/Security Adequacy Review from Sun
- Citigal's architectural risk analysis process.

2.2.2 Threat Modelling

Threat modelling is another important activity carried out at the design phase to describe threats to the software application in order to provide a more accurate sense of its security (Agarwal, 2006). Threat modelling is an engineering technique that can be used to identify threats, attacks, vulnerabilities and countermeasures that could affect a software system (Meier, Mackman and Wastell, 2005). This allows for the anticipation of attacks by understanding how a malicious attacker chooses targets, locates entry points and conducts attacks (Redwine and Davies, et al., 2004). Threat modelling addresses threats that have the ability to cause maximum damage to a software application.

A structured method for threat modelling has been defined by Microsoft and this consists of the following steps.

- **Identify security objectives:** This gives the software developers an idea of the business risk decision that needs to be taken thereby helping them to focus the threat modelling activity into building the necessary security control and also determines how much effort is to be spent on doing this.
- **Create an application overview:** This is done by surveying the software system's architecture and design documentation in order to identify the software system's characteristics and actors such as components, data flows and trust boundaries, which in turn would help to identify relevant threats.
- **Decompose the application:** This involves the decomposition of the software architecture in order to obtain a detailed understanding of the mechanics of the software. This would help to further identify the features and modules of the system that needs to be evaluated for security impact.

- **Identify Threats:** All known threats for the software system are identified during this step. Threats identified could then be categorized using the Microsoft threat classification scheme: STRIDE (Spoofing, Tampering, Repudiation, Information disclosure, Denial of service and Elevation of privilege).
- **Identify the vulnerabilities:** Following the identification of known threats to the software system, possible security weaknesses of the system are identified and these can also be categorized using DREAD. DREAD is Microsoft's classification model for quantifying, comparing and prioritizing the amount of risk by each evaluated threat (OWASP, 2008) and the acronym stands for Damage potential, Reproducibility, Exploitability, Affected users and Discoverability.

The figure below shows the five major threat modelling steps discussed above.

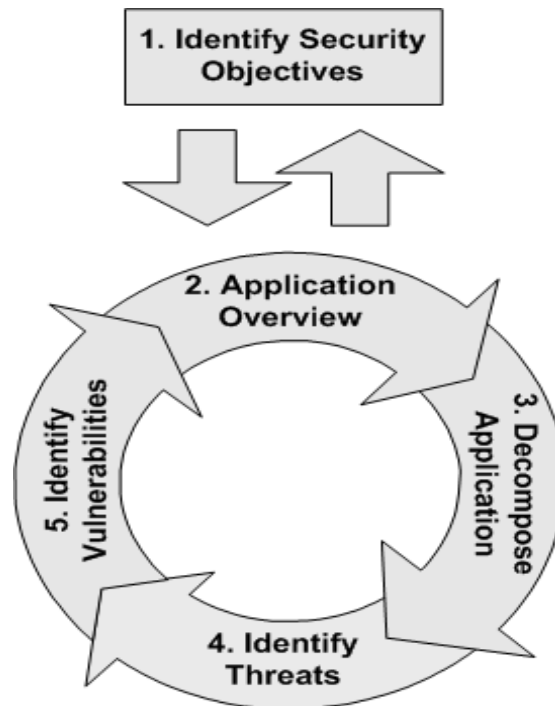


Figure 2. 1: Microsoft Threat Modelling (Meier, Mackman, and Wastell, 2005)

2.2.3 Dynamic Software Architecture Slicing (DSAS)

Kim T. et al. introduced the notion of dynamic software architecture slicing (DSAS) through which software architecture can be analysed. “A dynamic software architecture slice represents the run-time behaviour of those parts of the software architecture that are selected according to a particular slicing criterion such as a set of resources and events” (Kim et al., 2000). DSAS is used to decompose software architecture based on a slicing criterion. “A slicing criterion provides the basic information, such as the initial values and conditions for the ADL execuTable, an event to be observed, and occurrence counter of the event”.

To begin with, the software architecture is designed using any ADL (architecture description language) and then implemented by mapping the behavioural part of the design into program statements while retaining the structural properties. During run-time, the components and connector information is identified by the Forward Dynamic Slicer when it reads the ADL source code from the slicing criterion it receives as input, and executes the ADL execuTable to generate a set of partially ordered events. The events relevant to the slicing criterion are then filtered and passed to the Forward Architecture Slicer which examines the components and ports based on the given slicing criterion in order to compute an architecture slice dynamically. When the slicing criterion is satisfied the slice computed up to the event of interest is said to be the ‘forward dynamic software architecture slice’.

The main benefit of this approach is that software engineers are able to examine the behaviour of parts of their software architecture during run time. However, the trade-off of this approach is that it requires the software to be implemented first of all because the events examined to compute the architecture slice dynamically are generated while the Forward Dynamic Slicer executes the ADL execuTable.

2.2.4 Attack Trees and other related techniques

This is used to characterize system security by modelling the decision-making process of attackers. In this technique, attack against a system is represented in a tree structure in which the root of the tree represents the goal of an attacker. The nodes in the tree represent the different types of action an attacker can take on or outside the software system to accomplish his goal, which may be in the form of bribes or threats (Ralston, Graham and Hieb, 2007), (Gegick and Williams, 2006). “Attack trees are used for risk analysis, to answer questions about the system’s security, to capture security knowledge in a reusable way, and to design, implement, and test countermeasures to attacks” (Redwine and Davis, et al., 2004).

Attack nets is a similar approach, which includes “places” analogous to the nodes in an attack tree to indicate the state of an attack. Events required to move from one place to the other are captured in transitions and arcs. Arcs connect places and transitions indicate the path an attacker takes. Therefore as with attack trees, attack nets also show possible attack scenarios to a software system and they are used for vulnerability assessment in software designs (Gegick and Williams, 2006).

Another related approach is the vulnerability tree, which is a hierarchy tree constructed on the basis of how one vulnerability relates to another and the steps an attacker has to take to reach the top of the tree (Ralston, Graham and Hieb, 2007). Vulnerability trees also help in the

analysis of different possible attack scenarios that an attacker can undertake to exploit vulnerability.

Mouratidis et al. (2007) also propose a scenario-based approach called Security Attack Testing (SAT) for testing the security of a software system during design time. To achieve this, two sets of scenarios (dependency and security attack) are identified and constructed. Security test cases are then defined from the scenarios to test the software design against potential attacks to the software system. Essentially SAT is used to identify the goals and intentions of possible attackers based on possible attack scenarios. Software engineers are able to evaluate their software design when the attack scenarios identified are applied to investigate how the system developed will behave when under such attacks. From this, software engineers better understand how the system can be attacked and also why an attacker may want to attack the system. Armed with this knowledge, necessary steps can be taken to secure the software with capabilities that will help in mitigating such attacks.

Gegick and Williams (2006) have also proposed regular expression-based attack patterns which help in indicating the sequential events that occur during an attack. The attack patterns are based on the software components involved in an attack and are used for identifying vulnerabilities in software design. It comprises an attack library of abstraction that can be used by software engineers conducting Security Analysis for Existing Threats (SAFE-T) to match their system design. The attack patterns are based on a set of components that have been observed in a vulnerability that has been analysed and represented as a sequence of events during an attack. For instance the attack patterns begins with an event represented by the component used to trigger the attack, followed by successive events in the attack path and terminated by the threat target, which is the final objective of the attacker. The following is an example of a regular expression-based attack pattern, by the authors:

(Client +)(Server +)(LogFile +)(HardDrive +)

This attack pattern consists of four components i.e. Client, Server, LogFile and HardDrive. The authors state that the attack pattern can be read as “a series of Client (the start component) requests, followed by a series of Server actions, followed by a series of log updates to the LogFile, followed by a series of disk writes to the HardDrive (the threat target). The access log records an entry for each request and if enough requests are made, then the hard drive can be consumed by the access log file”. The figure below is a sample of a system design used to illustrate the attack pattern above by the authors.

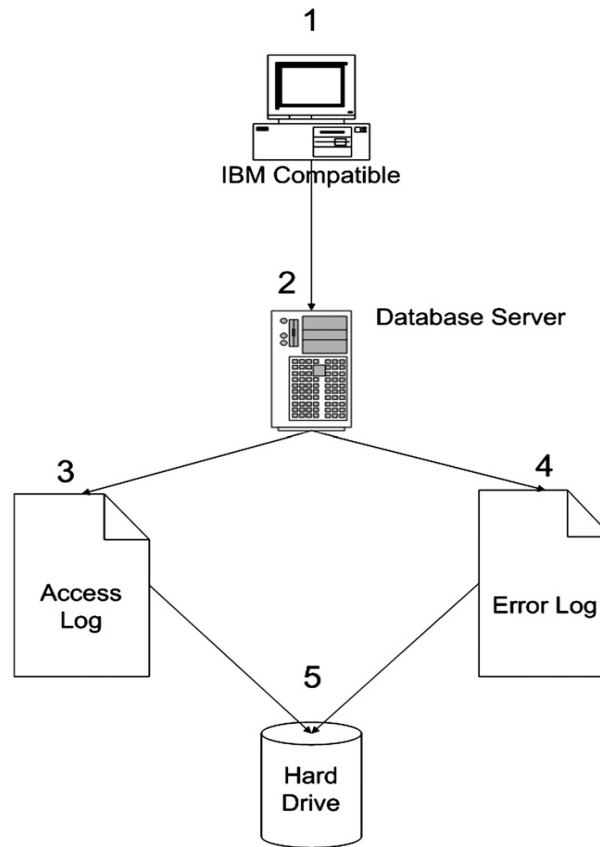


Figure 2. 2: Sample of System Design (Gegick and Williams, 2006)

Using the numbered components in the figure above the authors identified two attack paths corresponding to (Client *) (Server *) (LogFile *) (HardDrive *), these are 1-2-3-5 and 1-2-4-5. The authors affirm that when software developers are able to match the attack pattern to their software design, this would enable them to obtain the graphical representation of the attack path and the vulnerability. An occurrence of a match indicates that the vulnerability may exist in the system being analysed and therefore helps in integrating effective countermeasures before coding starts. Another advantage of this approach is that it can be easily adapted by developers who are novices on security unlike other approaches discussed above, which would need involvement of security experts. However, this approach can generate false-positive results when it is used.

The attack patterns were compared to taxonomies provided by Hoglund and McGraw, Landwehr et al. and Krsul. The authors reported that the attack patterns mapped to 62.2% of vulnerabilities abstracted by Hoglund and McGraw, 100% to that of Landwehr and 66% to that of Krsul.

2.2.5 Security Patterns

Several works have been done based on security patterns since the pioneer work of Yoder et.al, 1997 who applied design patterns to specific security issues. Different security patterns have been developed by many authors in different context which gave raise to different definitions on security pattern (Halkidis, et.al, 2006). However, most authors define security patterns as patterns describing particular recurring security problem in specific context and presents a well-proven solution to it (Wiesauer and Sametinger, 2009, Laverdiere et.al, 2006, Kiiski, 2007, Kienzle and Elder, 2002, Blakley, et.al, 2004).

These are design patterns that encapsulate security expertise solutions to recurring security problems that are applied to software designs to achieve security goals, such as availability, confidentiality or integrity (Agarwal, 2006) Through the use of security patterns, software developers are able to apply expertise worked solutions from security experts on their software design. In this way, developers are able to understand the strength and weaknesses of various approaches and make informed trade-off decisions on their design. Kienzle and Elder (2002) describe the objectives of security patterns as:

- A means of bridging the gap between software developers and security experts.
- A means of capturing security expertise solutions in form of worked-out solutions to recurring problems
- Intended to be used and understood by software developers who are not security professionals
- Intended to be constructive and educational as it tries to provide constructive assistance to software developers in the form of worked-out solutions (and not a checklist of what not to do) and also provide the guidance to apply the solutions properly

From objectives above, it can be noticed that the benefits of security patterns is that it provides an effective way for software developers who are not expert in security to learn from security experts. Since security patterns documents proven solutions to recurring problems in a well-structured manner that is familiar and easily understood by software developers it also enhances reusability of the patterns (Hafiz and Johnson, 2006). Therefore by using security pattern, software developers who are not expert in security able to expand their security focus from low level implementation to high level architectures (Schumacher, et.al, 2006)

In previous research, many authors describe security patterns for different purposes. This includes security patterns for web applications (Steel, et.al, 2005, Kienzle and Elder, 2003), security patterns for mobile Java code (Mahmoud, 2000), security patterns for agents systems (Mouratidis, et.al, 2003), Security patterns for Voice over IP (VoIP) (Fernandez, et.al, 2007) and security pattern for capturing encryption-based access control to sensor data (Cuevas, et.al, 2008). To enable software developers to choose the appropriate security pattern addressing the security risks in their designs, several authors have proposed different classification scheme for security patterns. This include classification based on applicability (Blakley, et.al, 2004), classification based on product and process (Kienzle and Elder, 2003) classification based on logical tiers (Steel, et.al, 2005), classification based on application domain (Bunke, et.al, 2011)

classification based on security concepts (Hafiz and Johnson, 2006), classification based on system viewpoints and interrogatives (Zachman, 1987), classification based on confidentiality, integrity and availability (CIA) model (Hafiz and Johnson, 2006) and classification based on attack patterns (Wiesauer and Sametinger, 2009)

For the purpose of this research work, further investigation was done on research work carried out by Wiesauer and Sametinger (2009) in chapter four. The motivation behind this research is to provide a new taxonomy for security design patterns that will enable software developers who are not necessarily expert in software security to easily select and apply them to their software design. The authors argued that because software developers without experience in security will not be able to apply security design patterns in a correct and effective manner, there is a need for pattern selection criteria. After identifying existing taxonomies and their drawbacks the authors proposed a new taxonomy based on attack patterns that will enable software developers to select appropriate security patterns according to possible attacks.

Halkidis, S.T. et al. (2006) conducted a qualitative analysis of the features of the security patterns developed by the Open Group Security Forum to investigate how they conform to three main sets of criteria and how they guide the software to be designed. The three criteria included:

- Guidelines by Viega and McGraw on how to build secure software. These guidelines are the design principles discussed below.
- A guideline based on software hole categories that enable software to be exploited by attackers as described by Viega and McGraw. The security vulnerability used under this criterion includes buffer overflow, poor access control and race condition.
- How well a specific security pattern can respond to different categories of attacks as identified by Howard and LeBlanc in the STRIDE model.

Their findings showed that not all the security patterns met all the criteria from the three sets of criteria used for the analysis. Therefore, as no security pattern has all the desired characteristics, it has been suggested that developers would need a good combination of the security patterns when designing software in order to make it secure.

However, combining security patterns to secure software design can also lead to other security holes. There can be inconsistencies among the security patterns, which may cause problems in the design when multiple security patterns are used to provide a level of security in software design. While each security pattern addresses a specific security problem, inconsistencies between patterns mean that security properties may no longer hold when they are combined and used to secure software designs (Dong J. et al., 2009).

2.2.6 Secure Design Principles

As many software developers often lack the much-needed security experience to develop secure software, secure design principles, such as the security patterns discussed above, bridge the gap between security experts and developers by offering guidelines that can help

developers build more secure software. The principles address practices that can be applied to architectural decisions and are recommended regardless of the language in which the software is to be written or the platform in which it would run (Barnum and Gegick, 2005).

The design principles of software security include:

- Securing the weakest link
- Defence in depth
- Failing Securely
- Least Privilege
- Separation of privilege
- Economy mechanism
- Least-Common Mechanism
- Reluctant to trust
- Never assume that your secrets are safe
- Complete Mediation
- Psychological Acceptability
- Promoting privacy.

To apply the principles, Over (2002), states that developers need:

- A supportive environment and infrastructure
- An operational process to put the principles into practice
- A measurement system to manage and control the result.

It should be noted that in many cases, software developers find these principles conflicting one another and this forces them to make trade-off decisions on their software development (Barnum and Gegick, 2005). As a result, software developers sometimes find these principles hard to follow and are not convinced of the benefits of disciplined software engineering methods (Over, 2002).

2.2.7 Defect Prevention and Reduction

As software riddled with defects poses a great security risk today, it has become increasingly important to prevent the defects from being introduced into the software or to reduce the number of defects to the minimum. As discussed earlier, security flaws are design errors that allow hackers, criminals or terrorists to obtain unauthorized access or use the software system (Humphrey, 2004). It is important to note here that a piece of software riddled with security defects may still function properly, since many of these defects do not cause functional problems. As a result, focusing on functional defects alone during software development would not be sufficient in dealing with potential security flaws in the system. Examples of software engineering practices which focus on reduction and prevention of overall software design and implementation defects include the Team Software Process (TSP) and Correction by Construction (CbyC). The approach taken by these practices to reduce software defects especially at the design phase of SDLC is discussed briefly below.

2.2.7.1 The Team Software Process (TSP)

TSP was developed by the Software Engineering Institute (SEI) as a set of defined and measured best practices for use by individual software developers and software development teams. TSP was designed as an operational process to support the establishment of good principles of software engineering (Over, 2002). TSP offers an integrated package which includes best practices for developing secure software. This includes team project management, risk management, process management, product quality management and software metrics.

Teams using TSP are first trained in the Personal Software Process (PSP) which enables the software developers to acquire the necessary skills of software engineering practice and also appreciate its benefits in developing secure software applications. According to Noopur Davis of SEI software development, teams using the TSP:

1. Manage defects throughout the software development lifecycle
 - a) Defect prevention so specification, design, and implementation defects are not introduced to begin with
 - b) Defect removal as soon as possible after defect injection
1. Control the process through measurement and quality management
2. Monitor the process
3. Use predictive measures for remaining defects

TSP does not only offer a process solution for producing near defect-free software applications, it also includes training for software developers in security issues, such as common causes of vulnerabilities, security-orientated design methods, secure coding and security testing especially with the TSP for Secure Software Development (TSP-Secure). This is a variant of TSP that augments TSP with security practices throughout SDLC for producing secure software. In their report on Security in Software Lifecycle Goertzel et al. (2006) states that TSP-Secure provides techniques and practices for:

- Vulnerability analysis by defect type
- Establishing predictive process metrics and checkpoints
- Quality management for secure programming
- Design patterns for common vulnerabilities
- Security verification
- Removal of vulnerabilities from legacy software.

The TSP-Secure research objective is to reduce or eliminate software vulnerabilities that result from software design and implementation defects, and to provide the capability to predict the likelihood of latent security defects in delivered software (Davis, 2005).

To archive this, the TSP-Secure quality management strategy is to have multiple defect removal points in the SDLC. This increases the likelihood of finding software defects from the time the defects are introduced. In this way defects detected can be easily fixed and their root causes investigated and addressed. Davis (2005), states that “each defect removal activity can be thought of as a filter that removes some percentage of defects that can lead to vulnerabilities from the software product”. Therefore with more defect removal filters in SDLC, defects leading to security holes in the released software products will be drastically reduced. Also, as defects are measured early during SDLC, software development organizations can save money by taking corrective action early in the SDLC.

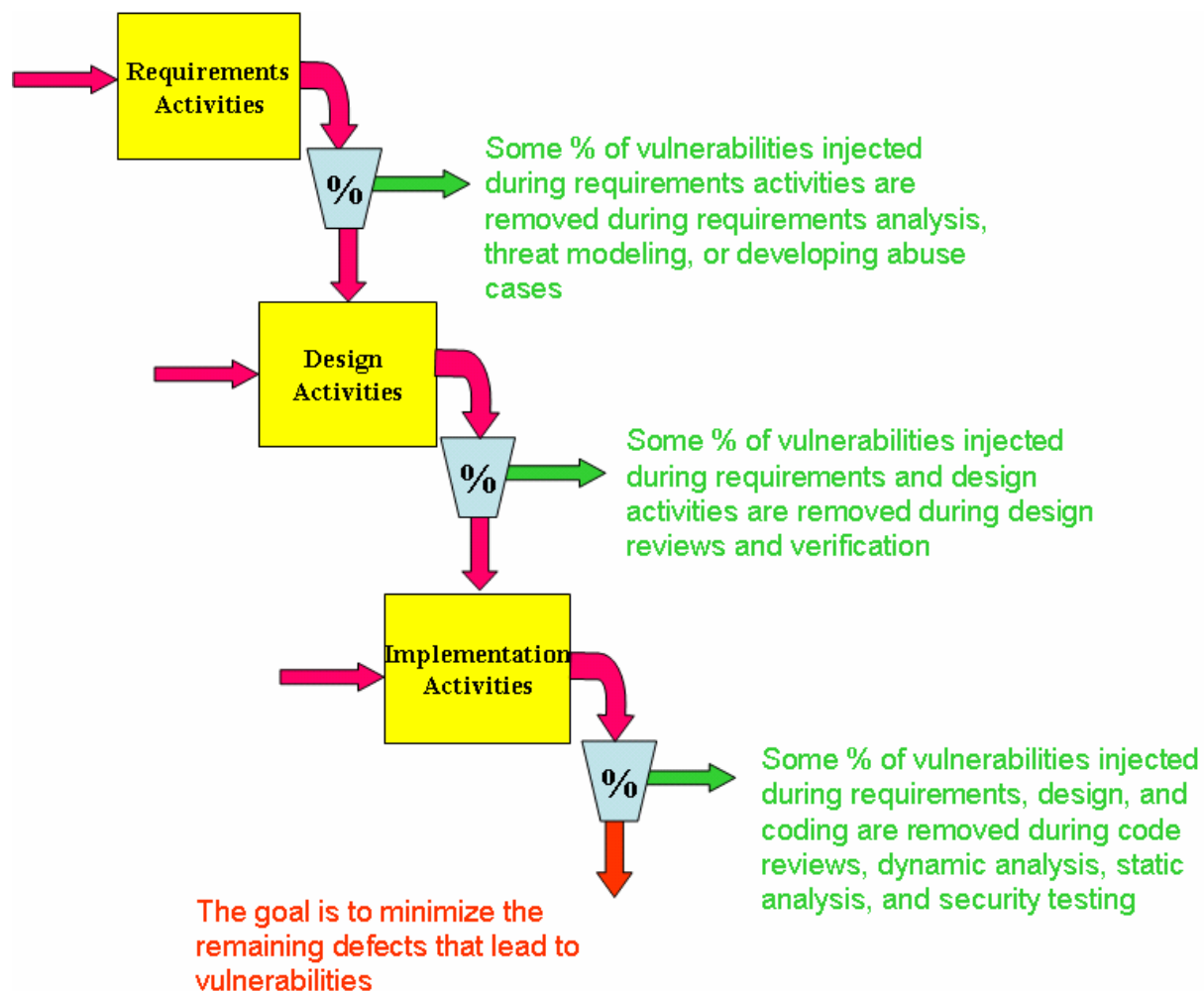


Figure 2. 3: TSP-Secure Methodology for Defect Removal (Davis, 2005)

TSP has been used by many organizations and it has helped to produce near defect-free software applications. It is reported in a study of 20 projects in 13 organizations that teams

using TSP-produced software have an average of 0.06 delivered design and implementation defects per thousand lines of new and changed code produced (Redwine and Davis, et al., 2004). Table 2.1 below shows the quality performance of these projects compared to other typical software projects.

Table2. 1 Quality of Performance of TSP Projects (Redwine and Davis, et al., 2004)

Measure	TSP Projects Average Range	Typical Projects Average
System test defects (design and implementation defects discovered during system test, per thousand lines of new and changed code produced)	0.4 0 to 0.9	2 to 15
Delivered defects (design and implementation defects discovered after delivery, per thousand lines of new and changed code produced)	0.06 0 to 0.2	1 to 7
System test effort (% of total effort of development teams)	4% 2% to 7%	40%
System test schedule (% of total duration for product development)	18% 8% to 25%	40%
Duration of system test (days/KLOC, or days to test 1000 lines of new and changed code produced)	0.5 0.2 to 0.8	--

2.2.7.2 Correctness by Construction

CbyC was developed by Praxis High Integrity System Limited in the UK and it has been used for over fifteen years to develop low-defect mission-critical software applications. It combines the best of both formal methods and agile development methodology (precise notation and incremental development respectively) (Amey, 2006). CbyC has evolved over the years and it is now applied all through SDLC from validation of the concepts of operation to preserving properties during long-term maintenance (Croxford and Chapman, 2005). The underlying principles of CbyC are:

- Do not introduce errors in the first place.
- Remove any errors as close as possible to the point that they are introduced.
- Generate evidence of fitness for purpose throughout the SDLC as a natural by-product.

In contrast to build and debug (the major way software products are developed today), CbyC aims to produce software products that are initially correct. With this in place, testing is no longer a point where debugging begins but a point where correct functionality of the software product is demonstrated. Amey (2006) describes CbyC as a natural fit to the goals of building security in security practices and states that CbyC “emphasizes the need to ensure that a system is developed integrating required properties rather than just retrospectively examined for those required properties.”

CbyC achieves its goals of not introducing any defects and making it easy for it to be detected and removed early by following the techniques below. These techniques form the CbyC process.

- **Use of Sound notation.** Sound formal notations are used to write the specification. This removes ambiguity, thus making it difficult for errors to be introduced.
- **Use of Strong Validation.** With the use of formal notation, carrying out proofs of formal specification and static code analysis using strong, tool supported methods is made possible.
- **Incremental Development.** This involves building the software in small increments and making sure that each increment behaves correctly.
- **Avoidance of Repetition.** A good example of how CbyC avoids repetition as described by Amey (2006) is the separation of software specification from high-level design. While software specification describes what the software will do, high-level design describes how the software will be structured and designed to meet requirements such as security safety and performance. Thus, the design does not repeat any information from the specification.
- **Striving for Simplicity.** To ensure that software that is easily validated is designed and produced, the process of verification is simplified and the code is kept simple and directly traceable to the specification.
- **Managing Risk.** CbyC tackles the most complex and least understood areas first when faced with a complex task as this is where risks and potential bugs are hidden.
- **Think hard.** This involves thinking hard about the real objectives of the software product to be developed and using the right tool for developing it.

As mentioned earlier, CbyC incorporates formal (mathematical) methods into the overall process of early verification and defect removal all through SDLC (Redwine and Davis et al., 2004). At the design stage the formal methods and notations are used to specify the behaviour of the software and to model its characteristics (Croxford, 2005). This forms high-level design, which gives a top-level description of the system's internal structure and also explains how the components work together. High-level design is validated by review and analysis to ensure correctness and consistency. For example, by using automated tools such as model checkers, a formal software design can be validated to ensure that it has desired properties such as freedom from deadlock (Hall and Chapman, 2004).

CbyC also contains a detailed design stage in which the set of software modules and processes is explicitly modelled and analysed. The module structure describes the software architecture and defines how functionality described in the specification and high-level design is allocated to each module. For secure systems, the system state and operations are categorized according to their impact on security with the aim of arriving at an architecture that minimizes and isolates

security-critical functions that reduce the cost and effort of the (possibly more rigorous) verification of those units (Hall and Chapman, 2004).

CbyC has proved to be very cost effective in developing software because errors are eliminated early during SDLC or not introduced in the first place. This subsequently reduces the amount of rework that would be needed later during software development. The use of formal methods helps CbyC to harness precision to every step of the development so that the system is more likely to meet the requirement and to work correctly when deployed. (Croxford, 2005) The Certification Authority system supporting the MULTOS multi-application smart card operating system was developed by Mondex International (now part of Mastercard) using CbyC. The system was delivered at a low operational defect rate of 0.04 defects/kloc (thousand lines of code)

CbyC has not been widely used. Hall and Chapman (2005) state ‘that this is because individuals and organizations do not believe that it is possible to develop software that is low defect’ Secondly, practical questions such as how to acquire the necessary capability or expertise and how to introduce changes necessary to make the improvement, need to be answered where the need for the improvement is acknowledged and considered achievable.

2.2.8 Formal Methods

“Formal methods are mathematically based techniques for the specification development and verification of software and hardware systems” (Hinchey et al., 2008). To overcome the problem of complex design errors, which when left undetected can lead to implementation defects that are difficult to detect and remove during testing, Howe (2005), argues that the industry needs to invest in solutions that apply formal methods in analysing software specification and design in order to reduce the number of defects before implementation starts. Recent advances in formal methods have also made verification of memory safety of concurrent systems possible (Hinchey et al., 2008). As a result, formal methods are being used to detect design errors relating to concurrency (Howe, 2005).

Hinchey et al. (2008) describes formal models as an engineering task that is best accomplished in the normal design discipline. With normal design, engineers are able to predict the behaviour of a system from its design before it is implemented. According to Hinchey et al. normal design embodies the accumulated knowledge of the models of the product in question and its environment that meets the desired level of dependability. This is implicitly handled in the configuration of a normal design and also considered in the checks and calculations mandated by the discipline of the normal design practice.

A software development process incorporating formal methods into the overall process of early verification and defects removal through all SDLC is CbyC (Redwine and Davis et al., 2004). CbyC has proved to be very cost effective in developing software because errors are eliminated early during SDLC or not introduced in the first place. This subsequently reduces the amount of rework that would be needed later during software development. Unfortunately, formal methods are not widely used and some software development organizations are reluctant to use them. Apart from this, formal methods require a mathematically meticulous way of judgment that many software developers are not used to (Redwine and Davis et al., 2004).

2.2.9 Secure Tropos

This is software development methodology that aims at integrating security into SDLC. Secure Tropos addresses the need to simultaneously analyse the technical and social dimensions of security to software systems. This methodology is based on Tropos methodology, which uses the *i** modeling framework. This framework proposes an agent-oriented approach to requirement engineering during SDLC by focusing on the intentional characteristic of the agent (Yu et al., 2007). The following concepts are used in the framework.

- **Actor:** This is an entity that has intentions and strategic goals within a system.
- **Goal:** This represents the actors' strategic interest. It is a condition or state of the world that the actor likes to achieve (Also referred to as 'hard goal').
- **Soft Goal:** This is used to capture non-functional requirements of the system to-be. It lacks the criteria for determining whether it is satisfied or not and therefore is subject to interpretation.
- **Task:** This is also referred to as 'plan' and represents a way of doing something.
- **Resource:** This represents a physical or informational entity that may serve a purpose or be required by an actor.
- **Social Dependencies:** These occur between actors in which one of the actors depends on the other to attain a goal, execute a task or deliver a resource. The depending actor is referred as the *dependor* and the actor depended upon is referred to as the *dependee*. The type of dependence between the two actors specifies the kind of agreement (*dependun*) between them.

The Secure Tropos methodology extends the Tropos methodology by integrating security into the concept described above and also by introducing new concepts. This includes the following.

- **Security Constraint:** This represents the restriction related to the security of the system (i.e. such as privacy, integrity, availability) that can influence the analysis and design of the system to-be during SDLC by restricting some alternative design solutions, by conflicting some of the requirements of the system or by refining some of the system's objectives.
- **Secure Entity:** This refers to any plan or goal relating to the security of the system.
- **Secure Goal:** This represents the strategic interest of an actor in relation to security. This is introduced in order to achieve the security constraint imposed on an actor or the security constraint imposed on in the system.
- **Secure Plan:** This represents a particular way of satisfying a secure goal.

- **Secure Dependency:** This introduces security constraints that both the *depender* and the *dependee* must agree to fulfill in order to satisfy the dependency.

The Secure Tropos process involves ‘analyzing the security needs of the stakeholders and the system in terms of security constraints imposed on the stakeholders and the system, identifying secure entities that guarantee the satisfaction of the security constraints, and assigning capabilities to the system to help towards the satisfaction of the secure entities’ (Mouratidis et al., 2007). The Secure Tropos methodology has four phases, which are:

- 1) **Early Requirement Phase:** During this phase the activities below are carried out.
 - A security reference diagram is constructed.
 - Security constraints are imposed on the stakeholders of the system.
 - Secure goals and entities are added to corresponding actors in order to satisfy the security constraints.
- 2) **Late Requirement Phase:** The activities carried during this phase include the following.
 - Security constraint are imposed on the system under development in reference to the security reference diagram.
 - Security constraints are further analyzed.
 - Identification of security goals and entities that are necessary for the system to guarantee security.
- 3) **Architectural Design phase:** This stage includes the activities below.
 - Analysis of security constraints and secure entities that may be introduced by new actors.
 - Definition of the architectural style of the system with regards to security requirements.
 - Transformation of the security requirements of the system into a design with aid of security patterns.
 - Identification of the agents of the system and their capabilities.
- 4) **Detailed Design:** During this phase the components identified in the previous phase are designed with the aid of the Agent Unified Modeling Language (AUML).

2.2.10 Alloy

Alloy is a tool used for the analysis of software design. With the high-level coding notation of a software design, the analysis tool is used to check billions of possible executions of the system for unusual conditions and constraints that will cause the system to behave in an unexpected way. This helps in detecting the design flaws before the software is coded and therefore results in a more reliable and robust software design.

Alloy is built upon two elements that help in making software designs more robust. One of the elements is the new language that helps to reveal the structure and behaviour of the software

design and the second element is an automated analyser that incorporates the SAT (satisfiability) solver to analyse the multitude of possible scenarios of the software during execution. Alloy works by trying to find solutions to software design puzzles that meet all the good constraints that could occur during the execution of the software as well as the bad constraints that may make the software behave unexpectedly during execution (Jackson, 2006). The solution (also called the 'counterexample') if found would reveal flaws in the design which would need to be fixed.

To use Alloy to analyse a software design the first step taken is to create the model of the design that specifies the moving parts and specific behaviours (both desired and undesired) of the system and its components. The software engineer will have to write down the definitions of the various types of objects in the design and then group the objects with similar structure and behaviour into mathematical sets. Following this, the facts that constrain these sets and their relation are identified. These facts include the mechanism of the software system and assumptions about other components, such as how human users are expected to behave. Some of the facts may be simple assumptions and may reflect the design itself. From the facts, assertions are made specifying that the system can never get into certain undesirable states and that specific bad sequences of events can never occur (Jackson, 2006).

Alloy uses the SAT solver to search for counterexamples, i.e. possible scenarios of the software system that are permitted by its design but violate the stated assertions. Alloy does this by constructing situations that satisfy the facts but go against a stated assertion that will make the system behave in an unacceptable way. The discovery of such scenarios would reveal the flaws in the software design. It is important to note that the essence of the software design is captured by an abstraction made up of the declarations of the sets and relations, the facts and the stated assertions that all make explicit the limitation of the software design. This compels the software engineer to find which abstractions will work best for the system.

So far, Alloy has been used to uncover flaws in some published software designs such as "a key management protocol that was supposed to enforce special-access rules based on membership in a group but turned out to grant access to former members who should have been rejected." (Jackson, 2006) Though tools such as Alloy are yet to be widely used in the software industry, their use during software development will help software developers to better evaluate their software design thereby producing more reliable and robust software systems.

2.2.11 Tools for evaluating security in Software design

Security related activities such as threat modelling and risk analysis has historically been in the domain of security experts. Over the years this has created a knowledge gap between the

software developers who are trained to think of functions and features of their product and its on-schedule delivery and the security expert who focus on observing system intrusions, dealing with malicious attackers and studying software vulnerabilities (Kenneth, Wyk and McGraw, 2006). To reduce this knowledge gap some security tools has been developed in the recent years which enable software developers to scrutinize their software design and identify security flaws in a similar way as a security expert (Adebiyi, et.al, 2012).

Microsoft developed two of these security tools. The first that was release for public use is the Threat Analysis and Modelling (TAM) tool. This was developed by the Microsoft Application Consulting; Engineering (ACE) team with the aim of enabling non-security expert software developers to use already known data and specific line of business application requirement and architecture to carry out threat modelling in an asset-centric approach. With this tool software developers can focus on protecting the assets within their application by identifying associated threats and counter-measures when it's being designed. This information enables the software developers to understand and manage business risks in their application early in the SDLC.

Following the release of TAM tool, Microsoft also released the SDL Threat Modelling Tool. This tool had been used extensively by Microsoft for threat modelling internally prior to its release. SDL Threat Modelling tool is a core element in the design phase of Microsoft Security Development Lifecycle which helps software developers to analyse their software designs prior to its implementation. According to Microsoft, this tool was not developed for security experts but for software developers to aid the creation and analysis of threat models (Microsoft, 2011). In contrast to TAM tool, SDL Threat Modelling tool builds on well-known development activities such as the use of data flow diagram (DFD) for drawing the architecture of the software being designed. Thus, following a software-centric approach, threat modelling with this tool focuses on the software and the analysis of its design (Swigart and Campbell, 2008).

While these tools have lots of useful features that enable software developers to do threat modelling easily, they have a few draw backs. Firstly, the quality of report generated by the tools is still limited by the knowledge of the software developer creating the threat model. Secondly, software developers require the understanding and interpretation of the extensive list of threats identified by the tools. This may become a daunting task especially when the threats are not prioritized as the case is with the use of SDL Threat Modelling Tool. Thirdly the process of threat modelling can increasingly become complex while using the tools due to factors such as number of developers involved in the threat modelling process, the nature of DFD created and potential stakeholders)(Mockel and Abdallah, 2011) (Berg, 2010).

There is now a range of security tools from open source with similar threat modelling approach like that of Microsoft threat modelling tools such as SeaMonster, TRIKE and Coras, which use

techniques that software developers are familiar with for the identification and mitigation of threats. There are other threat modelling approaches based on standards such as the Risk Analysis Toolkit based on ISO 1799 which generates security policies based on question and answers (Ricard, 2011) and other open security tools like the Common Vulnerability Scoring System (CVSS) that is designed to for rating IT vulnerabilities (CVSS-SIG, 2011).

2.2.12 Network Security

Network security is the process of taking both physical and software measures to protect networks and their services from unauthorized modification, destruction or disclosure and the provision of assurance that the network would perform its critical functions correctly without any harmful side effects (Gregory, 2001). Some of the network security tools used for protecting various networks includes firewalls, anti-virus, anti-spyware, software intrusion detection and intrusion prevention systems, encryption, virtual private networks (VPN) and vulnerability scanners.

Today, the network security environment has become a mature field with the development of various tools and technologies for securing the network. It had also influenced the development of technologies such as application firewalls and encryption devices for securing software product in a reactive way. However, as deployed software now come under continuous attack, it is being argued that so much time, money and effort would not have been spent on network security if we didn't have bad software security (Viega and McGraw, 2002). This stems from the fact that building secure software is better than protecting a bad one. On this note, Gary McGraw of Citigal Inc stated that "trying to protect software from attack by filtering its input and constraining its behaviour in a post facto way (application security) is nowhere near as effective as designing software to withstand attack in the first place (software security)".

Therefore, while network security aims at protecting networks and its resources such as the software applications running within it, software security aims at integrating security into software during development in order to produce secured software applications that can withstand attack proactively in a hostile environment. Thus, network security could be viewed as providing security in depth to software applications through the use of network security tools to establish many layers of protection in the network environment whereby software security helps in providing security within software applications itself. Hence, network security and software security can both work together to provide an overall protection against malicious attackers within networks and software applications.

2.2.13 Application Security

With the increase of attacks targeting vulnerable software applications, the need for application security measures arose to augment network security measures as this could no

longer provide adequate security for software applications. Application security uses a combination of system engineering security practices, such as defence-in-depth (DiD) measures (e.g. protecting against malicious code, encryption, extensible markup language (XML) security gateways, locking down executables, sandboxing, application layer firewalls, policy enforcement) and secure configurations, with operational security practices, including patch management and vulnerability management to protect software application from attacks (Goertzel, et.al, 2007). Application Security protection measures are mostly defined at network and system architectural level instead of the individual software architectural level and these are implemented when the software application is deployed and fully operational (McGraw, 2002).

One of the benefits of application security is that it minimizes the exposure of vulnerable software to threats that could exploit them by using security practices such as patch management to reduce the number of vulnerabilities in the software application. Another benefit of application security is that it helps to specify trust boundaries which controls access to vulnerable software application and also provide safe execution environment through the use of boundary protection technologies (e.g. application firewall) that offer protection from attacks. This subsequently constrains interactions with the vulnerable software application, thus reducing its exposure to attackers.

In analogy to band aid that offers protection against the wound but does not remove the disease, some authors have been referred to application security as band aid security based on the benefits highlighted above. These authors argue that these benefits are in favour of application security as many software developers do not have the skill to integrate security into software during development. Furthermore, as banks and many other companies through mergers and acquisition take over software applications often infested with various vulnerabilities, it has been argued that application security provides the cheapest way of maintaining the application because most of these companies do not have the money or time to rebuild the applications (Hogland, 2002).

In a contrary view, application security has been described as protecting software applications in a reactive way by finding and fixing security problem only after they have been exploited. While application security deal with security problems under the band –aid security approach, i.e. addressing security symptoms such as stopping buffer overflows attacks by monitoring the application inputs or HTTP traffic on port 80, it ignores the root cause of the problem which is the vulnerability in the software itself. Therefore, some authors have argued that ‘in the fight for better software, treating the disease itself (poorly designed and implemented software) is better than taking an aspirin to stop the symptom’(McGraw, 2002) This is the software security approach and it involves software risk management, secure coding, design for security and security tests during software development. In support of this argument is the IEEE P1074 Standard for Developing Project Life Cycle Processes. According to Bar Biszick-Lockwood who headed the volunteer team who proposed this standard to IEEE, P1074 is the first IEEE software

process Standard to embed dedicated, mandatory, security-related activities in the software development life cycle.

2.3.0 Neural Networks

Neural Networks were inspired by the studies of the brain and nervous system in biological organisms. An artificial neural network consists of very large highly interconnected simple processing elements (called artificial neurons) which can demonstrate complex overall behaviour depending on the interconnected neurons and element parameters. The interconnected processing elements work together to solve specific problems through a learning process just like biological systems.

A typical neuron in the human brain that neural networks model, collects signals from others through a fine structure of projections called the dendrites. The neuron uses an axon (a long, thin stand which also splits into many branches) to send out spikes of electrical activity when it receives an excitatory input that is suitably larger than the input. Another structure, called a synapse, at the end of each branch of the axon converts the activity of the axon into electrical effects that either excite or inhibit connecting neurons.

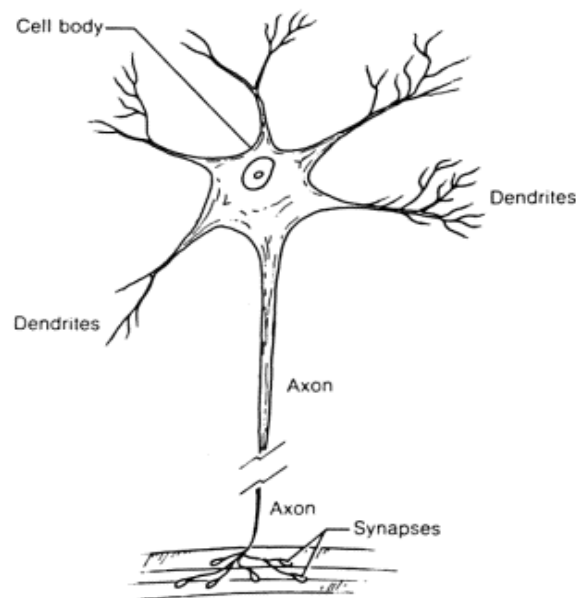


Figure 2. 4: Structure of a Neuron (Turchin, 1977)

Artificial Neural Networks (ANN) have been used in various applications to extract and identify trends that are too complex to observe by computer techniques or by humans because of their ability to obtain meaning from complicated or incomplete data. They are used to infer functions from various observations, which can be used to provide projections – given new situations of interest – and answer “what if” questions. Other advantages as stated by Stergiou and Siganos, 1997 include:

- 1) **Adaptive learning:** An ability to learn how to do tasks based on the data given for training or initial experience.
- 2) **Self-Organisation:** An ANN can create its own organisation or representation of the information it receives during learning time.
- 3) **Real Time Operation:** ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured that take advantage of this capability.
- 4) **Fault Tolerance via Redundant Information Coding:** Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage.

2.3.1 Artificial Neural Network and Conventional computers

While conventional computers are good at calculating arithmetic very fast and doing precisely what they have been programmed to do, they are not good at:

- Dealing with noisy data from the environment
- Massive Parallelism
- Adapting to circumstances.

The reason for this is because conventional computers resolve problems using an algorithm approach that follows a set of instructions as specified by the programmer. Without these instructions, the computer will not be able to solve the problem. This factor restricts the problem-solving capability of conventional computers to well-understood problems and the methods of resolving them.

Unlike conventional computers, neural networks cannot be programmed to carry out specific tasks. They learn by example. Information is processed similarly to the way the human brain processes information through its large number of highly interconnected network of processing elements that work in parallel to resolve specific tasks.

Smith 2003, states that ANN can help especially in the following areas:

- Where an algorithmic solution cannot be formulated
- Where structure needs to be picked out from existing data
- Where there are lots of examples for the behaviour that is required.

Up till now neural networks have been used successfully in many disciplines for different applications. They have been used in the area of forecasting, predicting, data validation, marketing, risk management in many business organizations and also in the medical field. The area most relevant to this research where neural networks have been used is in the area of network security and application security. This is discussed below.

2.3.2 Neural Network Architectures

Neural network architectures are divided into two categories: supervised and unsupervised architectures. In the supervised architecture the neural network is trained to give desired outputs when inputs are fed into the network. Examples of supervised architecture include:

- **Feed-Forward Networks:** This network has only a one way connection between its input and output layer through which signals are allowed to travel. It is a straightforward network associating inputs with outputs. They are commonly used for pattern recognition, prediction and non-linear function fitting (Stergiou and Siganos, 1997) (MathWorks, 2011). Examples of feed-forward networks include perceptron networks, feed-forward back-propagation, linear networks, feed-forward input delay back-propagation and cascade-forward back-propagation.

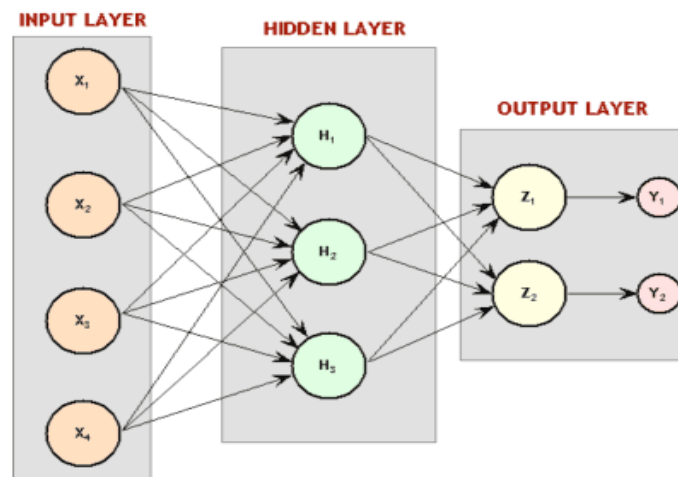


Figure 2. 5: Feedforward Network (Edward, 2008)

- **Feedback Networks.** These networks allow signals to travel in both directions, thereby introducing loops in the network. This makes the networks dynamic as their state keeps changing continuously until it reaches an equilibrium point. They are usually used for nonlinear dynamic modelling, control system application and time-series prediction.

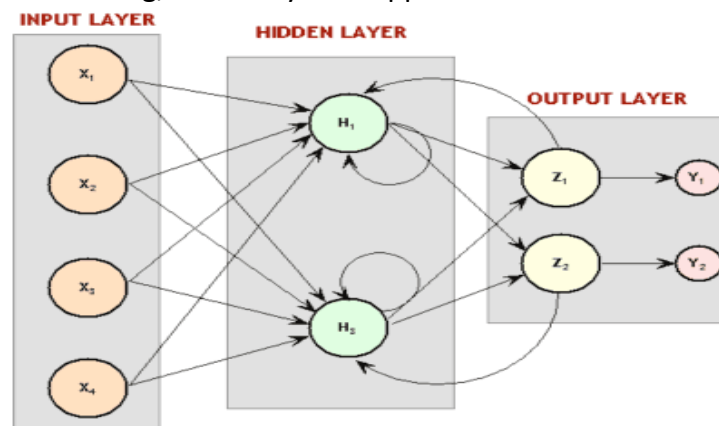


Figure 2. 6: Feedback Network (Edward, 2008)

- **Radial basis Network:** This is a network with only one hidden layer, which provides an alternative method for designing non-linear feed-forward networks (MathWorks, 2011) (Smith, 2003). The hidden layer has a receptive field with a centre in which outputs tails off as the input moves away from it.
- **Learning Vector Quantization (LVQ):** This network enables classification of patterns that are not linearly separable. Class boundaries and granularity of classification can be specified with LVQ.

In unsupervised architecture neural networks, no external teacher is involved. The neural network self-organises the data presented to it and detects their emergent collective properties (Stergiou and Siganos, 1997). Examples of unsupervised architecture include:

- **Competitive Layers:** This network recognises similar input vectors and groups them together. This eventually allows for the inputs to be automatically sorted into categories. Competitive layers are usually used for pattern and classification recognition.
- **Self-Organizing Maps:** This also groups inputs according to their similarity but differs from a competitive layer in that it is able to preserve its topology of the input vectors such that nearby inputs are assigned to nearby categories (MathWorks, 2011).

2.3.3 Applications of Neural Networks

Previous researches have shown how neural networks can be used in the area of network security as intrusion detection systems. In this area, neural networks are being used to decide when an attack is taking place against a computer system and this has offered possible solutions to some of the problems experienced by other present approaches to intrusion detection, which are rule-based systems (Cannady, 1998), (Ahmad, Swati and Mohsin, 2007).

Neural networks were proposed to recognize the distinctive characteristics of users of software systems and point out statistically significant variations from their recognized behaviour using data from the network environment even if the data is flawed or distorted. Since a network can come under several coordinated multiple attacks, neural networks have also helped to identify such attacks because of their ability to process data from a number of sources in a non-linear fashion (Ahmad, Swati and Mohsin, 2007). Furthermore, a neural network has the ability to detect novel attacks it has never been exposed to during its learning process. This is because by creating and refining abstractions from raw data, it learns not just what an attack is and what it's not, but what makes an attack an attack (McAvinney and Turner, 2005).

Ahmad, Swati and Mohsin (2007) proposed a neural network trained using a Resilient Back Propagation (RPROP) algorithm and this was tested against different types of network attack such as DOS and probing and the result gave an overall detection rate of 95.93% which is noted to be of better performance when compared to other neural network approaches to IDS. It is important to note here that although the neural network has been applied as a network security tool for detecting network attacks, it does not give a 100% detection of network

attacks as the result shows. However, it equally demonstrates the ability of neural networks to detect network attacks to a very reasonable degree.

Also, neural networks have been proposed for detecting computer viruses using statistical analysis approaches. The proposed neural network was designed to study the features of normal system activity and recognize statistical variations from the normal activity that may indicate the presence of a virus (Cannady, 1998).

In the area of software quality, neural networks have also been proposed for predicting the reliability of a software application by using the failure history of the software as their raw data to develop their own internal model of the failure process to predict the total number of faults to be detected at the end of a future test session of the software (Malaiya Y. K. et al., 1992). Similarly, Tamura, Yamada, and Kimura (2003), having noted the difficulty of assessing software reliability due to the increase in software complexity especially in a distributed environment proposed a software reliability assessment method based on a neural network model that takes into consideration the interactions of software components in the distributed environment. This method was compared to other software reliability growth models (SRGMs) using real software fault data to conduct a goodness of fit comparison and the result showed that this method gave the best fit to the data. Owing to the positive performance of neural networks in this area, it has been suggested as an alternative in modelling software reliability data (Ho, Xie and Goh, 2003).

Neural networks have also been applied in the area of cryptography. Karras and Zorkadis (2003) used neural networks for strengthening the existing traditional generators of random numbers used in many cryptographic protocols for secure communication systems. The use of the Overfitting Multilayer Perceptron (MPL) type of neural networks and recurrent ANN of the Hopfield type for generating pseudorandom numbers were the two types of neural network-based mechanisms proposed. Their performance was compared to the traditional generators and the results showed that ANN-based generators performed better in the statistical and non-predictability tests than the traditional generators. In another approach by Lian (2008), a neural network is used to construct a block cipher that has the ability to encrypt data with the control of its key. The neural network consists of the chaotic neuron layer and the linear neuron layer which helps it to realize data diffusion and confusion functionalities respectively and these are the essential properties for the operation of a secure cipher. As a result, it offers a higher security against select cipher attacks when compared to other existing ciphers and it is more suitable for encrypting images.

In the area of application security, a neural network has been proposed to solve the problems occurring in systems using a password Table and a verification Table for user authentication where an attacker can add a forged user-ID and password to the verification Table or replace

someone's password. This approach was proposed by Lin, Ou and Hwang (2005) using the characteristic memory of a back-propagation network to recall the password information generated and memorized during its training. This proposed method was tested using 200 pairs of usernames and passwords consisting of eight characters. In terms of the accuracy, the result of this proposed method showed that the output of the trained neural network was close to the expected output and, with regards to its performance, its time complexity was found to be 0. This is because it uses simple multiplication and addition to produce its results unlike the methods using encryption to store password information on the verification Table requiring exponential computing, which is more time consuming. However, it is noted that it takes a long time to train the network and that the proposed method does not protect against guess attacks in which an attacker can try all possible passwords until a match is found.

2.4. Summary of Chapter Two

Creating software applications that work and at the same time will continue to function correctly under malicious attack remain a very great challenge to the software industry. Many of the approaches used to integrate security into software application during the design phase SDLC has been explored in this chapter. While these approaches help towards developing a secure software application, the need for involvement of security experts and developers lack of experience in security was noted as factors affecting their effectiveness. The use of formal methods and security tools were also explored and their draw backs such as lack of adoption were highlighted. Network security and application security protects software applications by embracing standard approaches such as penetrate and patch, input filtering (i.e. blocking malicious inputs), monitoring programs as they run, enforcing software use policy with technology and providing value in a reactive way. While network security and application security offers protection to software applications after development, it was established that it is better to integrate security during software development. The background to neural networks was presented in this chapter and its applications especially in the area of network security and application security in technologies such as IDS, cryptography, anti-virus and user authentication techniques. In the next chapter the proposed neural network approach is presented and neural network model overview will be discussed.

Chapter 3. Software Design Evaluation by Neural Network

3.1. Introduction

Neural network is proposed as an evaluation tool in this research for evaluating software designs for security flaws. The evaluation by the proposed neural network tool is based on two previous research works by Michael Gegick and Laurie Williams (2006) and Wiesauer and Sametinger (2009) (See chapter 2 for further discussion). In the first research work, Gegick and Williams (2006) created regular expression based attack patterns which show sequence of events that could occur during an attack. These attack patterns are based on software components involved in attacks and are tailored towards identifying security flaws in system design. The authors argued that software engineers can conduct a Security Analysis for Existing Threats (SAFE-T) by matching the attack patterns to their system design. Their motivation for this approach is to enable software developers identify vulnerabilities in their system design before coding start as this is seen as a cheaper way of integrating security into software compared to when it is been retrofitted after development. In the second research work, Wiesauer and Sametinger (2009) proposed a new taxonomy for security design patterns based on attack patterns. The authors argue that there is a need for a selection criterion because software developers who are not experienced in security are unable to select and apply security pattern in a correct and effective manner. Therefore by using their proposed taxonomy the authors state that software developers can match identified attack patterns in their software designs to corresponding security design patterns that would provide the appropriate mitigation.

One of the limitations with some of the current approaches for integrating security into software design is the difficulty of getting software developers to think like attackers during the threat modelling process as this mind-set is not native to them (Swigart and Campbell, 2008). It has been suggested that software developers can instead look at the attack surface of their software design and think of how to build defences into their application (Swigart and Campbell, 2008). The proposed neural network tool achieves this by associating components in the design with attacks that can be performed on them when possible attack patterns are matched to the software design. This subsequently assists the software developers in addressing security defences needed to be put in place.

In this chapter, the model overview of this research work is presented. The components of the research model show the way through which the proposed neural network tool would be evaluating software design for security flaws based on the previous research works highlighted above. As a result, each of the components of the research model would also be discussed.

3.2. Model Overview

The figure below shows the model overview. This includes the input module, the neural network module and the output module. The input module consists of the system design, which is in form of scenarios that has been abstracted from a software design. The scenarios consist of the participating software components and actors in the design during each scenario. The neural network module consists of two neural networks. The first neural network is trained to recognise possible attacks from the set of scenarios presented to it by matching it to corresponding attack patterns proposed by Gegick and Williams (2006). The second neural network accepts the output of the first neural network (i.e. identified attack pattern) as input and suggests possible security design pattern that could mitigate the threat in the attack pattern identified in the first neural network. The output module simply shows the attack identified by the first neural network and the security pattern suggested by the second neural network that has been mapped to deal with the attack.

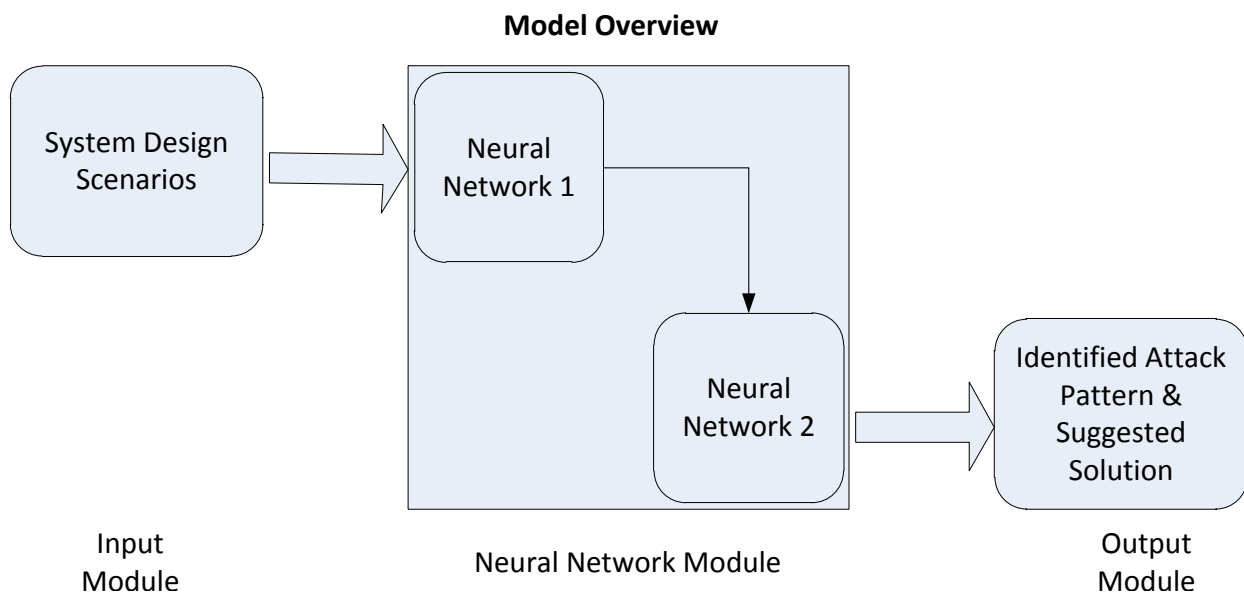


Figure 3. 1: Model Overview

3.3 The Input Module

The proposed Neural Network tool uses the abstract and match technique for identifying security flaws in software designs by matching possible attack pattern to the design. For the purpose of this research work, data was abstracted from the attack scenarios in the reported attacks in online vulnerability databases. As all the attacks considered were mainly caused by security flaws in the design of the reported software application, the abstracted attack scenarios were used as the software design in the input module of the research model because this provided information on the software design of the reported vulnerable software application. In a similar way, using well known approaches such as data flow diagrams (DFD)

and sequence diagrams, software developers are able to abstract information from scenarios in their software designs needed by the Neural Network tool for matching possible attack patterns to their design. In the following sections, the source of the attack scenarios and the attributes used to abstract the information needed to train the neural network are discussed.

3.3.1 Source of attack scenario

To generate the attack scenarios linking the software components and actors identified in the attack pattern, online vulnerability databases were used to identify attack scenarios corresponding to the attack pattern. Data of attack scenarios from the following online vulnerability databases were used in this research.

1. CVE Details
2. Security Tracker
3. Secunia
4. Security Focus
5. The Open Source Vulnerability database (OSVD)
6. IBM Internet Security Systems

For each of the attack scenarios, the online vulnerability databases gave various types of information. The variety of information given on the same attack scenarios provided comprehensive information about the attacks. Most of the online vulnerability databases gave a brief description of the attack as seen in the figure below. NoTable among other information provided is information about source of the attack (i.e. whether the attack is a remote attack or a local attack), the impact of the attack on confidentiality, integrity and availability of the reported software application, the vulnerability exploited in the attack, the CVSS score of the attack which indicated the severity rating of the attack, details of exploits code and level of access gained by the attacker. The figure below shows the information provided on the attack on webmail server from CVE details, security focus and security tracker online databases.

Vulnerability Details : [CVE-2003-1192](#)

Stack-based buffer overflow in IA WebMail Server 3.1.0 allows remote attackers to execute arbitrary code via a long GET request.


Publish Date : 2003-11-03 Last Update Date : 2008-09-05

[Collapse All](#) [Expand All](#) [Select](#) [Select&Copy](#) [▼ Scroll To](#) [▼ Comments](#) [▼ External Links](#)
[Click here](#) if you can't see the dropdown menus or if you want to expand them now

– CVSS Scores & Vulnerability Types

Cvss Score	10.0
Confidentiality Impact	Complete (There is total information disclosure, resulting in all system files being revealed.)
Integrity Impact	Complete (There is a total compromise of system integrity. There is a complete loss of system protection, resulting in the entire system being compromised.)
Availability Impact	Complete (There is a total shutdown of the affected resource. The attacker can render the resource completely unavailable.)
Access Complexity	Low (Specialized access conditions or extenuating circumstances do not exist. Very little knowledge or skill is required to exploit.)
Authentication	Not required (Authentication is not required to exploit the vulnerability.)
Gained Access	Admin
Vulnerability Type(s)	Execute Code Overflow
CWE ID	CWE id is not defined for this vulnerability

Figure 3. 2: Information on CVE Details database showing attack on webmail

 **SecurityFocus™**

Symantec Connect
A technical community for Symantec customers, end-users, developers, and partners.
[Join the conversation >](#)

[info](#) [discussion](#) [exploit](#) [solution](#) [references](#)

IA WebMail Server Long GET Request Buffer Overrun Vulnerability

Bugtraq ID:	8965
Class:	Boundary Condition Error
CVE:	
Remote:	Yes
Local:	No
Published:	Nov 03 2003 12:00AM
Updated:	Nov 03 2003 12:00AM
Credit:	The discovery of this vulnerability has been credited to Peter Winter-Smith.
Vulnerable:	True North Software IA WebMail Server 3.1 True North Software IA WebMail Server 3.0

Figure 3. 3: Information on Security Focus database showing attack on webmail

Category: [Application \(Web Server/CGI\)](#) > [IA WebMail Server](#)
Vendors: [True North Software](#)

IA WebMail Server Buffer Overflow in Processing HTTP Headers Lets Remote Users Execute Arbitrary Code

SecurityTracker Alert ID: 1008075
SecurityTracker URL: <http://securitytracker.com/id/1008075>
CVE Reference: [CVE-2003-1192](#) ([Links to External Site](#))
Updated: May 19 2008
Original Entry Date: Nov 3 2003
Impact: [Denial of service via network](#), [Execution of arbitrary code via network](#), [Root access via network](#), [User access via network](#)

Version(s): 3.1 and prior versions

Description: Peter Winter-Smith reported a buffer overflow vulnerability in the IAWebMail Server. A remote user can execute arbitrary code, potentially with System privileges.

A remote user can reportedly submit an HTTP GET request with 1044 bytes to cause the target server to execute arbitrary code. The code will run with the privileges of the IAWebMail Server process, which may be SYSTEM privileges on some systems (however, this is reportedly not the default configuration).

A remote user can also send a long string to the application to cause the service to consume 100% of system resources and eventually crash.

The vendor has reportedly been notified.

Impact: A remote user can cause the target service to execute arbitrary code.

Solution: No solution was available at the time of this entry.

Vendor URL: www.tnsoft.com/webmail.htm ([Links to External Site](#))

Cause: [Boundary error](#)

Underlying OS: [Windows \(Any\)](#)

Message History: None.

Figure 3. 4: Information on Security Tracker database showing attack on webmail.

3.3.2 Attack Attributes

For the purpose of training the neural network the following attributes were used to abstract the information needed from the attack scenarios.

- **The Attacker:** The information captured with this attribute is the capability of the attacker. This examines what level of access possessed when carrying out the attack. The following levels of the access were examined for each of the attack scenario.
 1. **No access:** This shows the attacker has no access when carrying out the attack. The attacker could escalate his privilege during the course of the attack. In some attack scenarios, no access is required by the attacker for carrying out the attack.
 2. **Read access:** This indicate where the attacker has a form of user account with access limited to reading or viewing information in an application
 3. **Write/Change access:** This also shows where the attack having a form of user account but with more privileges. In this case, the attacker is able to read and write. He is able to make changes or modify the data that he is able to access.
 4. **Admin access:** For this level of access, the attacker is able to read, write, make changes to data, create and delete user accounts. With this access, the attacker has full administrative right.

- **Source of attack:** This attack attributes captures the location of the attack during the attack. In particular, the attack scenarios are examined under this attribute to find out whether the attack was carried locally (i.e. internally) or remotely (i.e. externally)
- **Target of the attack:** This captures the system component that is targeted by the attacker. This could be a system resource such as memory, buffer or data that is stored in a database.
- **Attack vector:** This attributes captures the mechanism (i.e. software component) adopted by the attacker to carry out the attack. For instance, the webmail attack scenario in the figure above, the attacker uses a long Get Request to cause buffer overflow and also execute arbitrary code.
- **Attack type:** The security property of the application being attacked is captured under this attribute. The security property under attack could be:
 1. **Confidentiality:** This is when privacy in the application is compromised. For example, when information is disclosed to users who are not authorized to have access to it.
 2. **Availability:** This is when the attack is aimed at making services provided by the software system unavailable to valid users. For instance, in denial of service attack
 3. **Integrity:** Attack against this security property includes attacks where the data stored or processed by the targeted software application is maliciously modified.
- **Input Validation:** This attributes examines whether any validation is done on the input passed to the targeted software application before it is being processed. Attack scenarios are categorized into the following groups using this attributes
 1. **No Validation:** where no input validation was carried out on inputs
 2. **Partial Validation:** Where insufficient input validation was done on the inputs
 3. **All inputs validated:** Where all input was properly validated or where the attack was not associated to this security flaw.
- **Dependencies:** The interaction of the targeted software application with the users and other systems is analysed under this attributes. The intention here is find out whether any measure is implemented in the software application to make sure its dependency on users or other software system is secured. With this attributes, attack scenarios are examined under the following categories
 1. **None:** This is where no security measure is implemented before accepting any form of interaction from users or other software system. For example, attack scenarios where visitors to a website are allowed to post messages without any form of authentication
 2. **Authentication:** This examines whether authentication was carried out before interacting with users and other software system
 3. **Access Control:** In this category, the targeted software application in the attack scenario is examined for failure in restricting access to resources to unauthorized users

4. **Trust Boundary Defined:** This examines whether the targeted software application define a trust boundary between itself and other users or software system it is interacting with
 5. **Encryption:** For this category, the failure of the targeted software application to secure the communication between itself and other users or software systems when exchanging sensitive information is examined
- **Output encoding to external applications/services:** Attack scenarios are examined under this attributes to check whether the attack is associated with flaws due to failure of the targeted software application in properly verifying and encoding its outputs to other software systems. For instance, in SQL injection attacks, the targeted web application is checked for failure in sanitizing user supplied query before passing this to the database. Therefore, under this attributes, attack scenarios are examined to see if the flaws exploited in the attack is due to failure of the targeted software application to verify untrusted data using techniques such as escaping all inputs, use of parameterized interface or prepared statements.
 - **Authentication:** This attribute checks for failure of the targeted software application to properly handle account credentials safely or when the authentication is not enforced in the attack scenarios. Attack scenarios are examined under this attributes to see of the flaw is associated to any of the issues below include:
 1. Use of plaintext password
 2. Use of salted or harsh password
 3. Lack of account lock outs and time outs
 4. Session Management
 5. Credential management
 6. Lack of re-authentication for assessing sensitive data
 7. Lack of decision logs showing whether failure or success of authentication
 - **Access Control:** Failure in enforcing access control by the targeted software application is examined in the attack scenarios with this attribute. The attacked scenarios are analysed with flaws associated to the following:
 1. Data access authorization
 2. URL access authorization
 3. Service access authorization
 4. Function access authorization
 5. File access authorization
 6. Server side enforcement of access control
 7. Failure in checking all access path
 - **HTTP Security:** Attack Scenarios are examined for security flaws related to HTTP requests, headers, responses, cookies, logging and sessions with this attribute. The following issues are examined in the attack scenarios with this attribute:

1. Lack of validation when redirecting data
 2. Lack of defined HTTP request methods
 3. Failure to validate HTTP request and response
 4. Use of weak random token when processing or accessing sensitive data
 5. Lack of secure flag set up on cookies that contain sensitive data
- **Error handling and logging:** Attack scenarios are examined under this attributes for failure of the targeted application in safely handling error and security flaws in log management. Security flaws examined with this attributes include
 1. Display of sensitive data in error messages
 2. Lack of server side controlled logging
 3. Access not denied by default by the error handling logic
 4. Failure to log success or failure of security relevant log
 5. Failure to control access to log
 6. Lack of validation of untrusted data used in event logged

3.4. The Neural Network Module

The neural network module contains two neural networks. The first neural network is trained using the information from the input module to match possible attack pattern. The second neural network uses the information about the identified attack pattern to match possible security design patterns that can mitigate the threat in the attack. During the implementation of the two networks in this research, the neural networks were designed to work independently as two separate networks and not as a single system. The neural network module as shown in Figure 3.1 illustrates how the two networks relate to each other.

3.4.1 Neural Network I

The abstract and match technique has been used as a security identification technique for abstracting vulnerabilities that can be matched to different set of software systems. Attack trees and other related techniques use these techniques (see chapter 2 for further discussion) to abstract known vulnerabilities to a high level representation in a generalized form so that software developer can match the abstracted vulnerabilities in the same or different software systems they have been found originally (William and Gegick, 2006). Building on this approach, William and Gegick (2006), proposed the regularly expressed attack patterns for representing vulnerabilities in software design in a generic way (i.e. independent of any software application and programming language) so that this could be easily adopted by software developers for matching the attack patterns to their software design. Table 3.2 in section 5 shows the 53 regularly expressed attack patterns proposed by the authors.

Following a similar approach, the first neural network in the neural network module uses the information abstracted from software design scenarios in the input model to match the security

flaws in the scenarios to the regularly expressed attack patterns proposed by William and Gegick (2006). Figure 3.5 shows the steps taken to achieve this. The output of the first neural network (i.e. the identified attack pattern) is used as an input for the second neural network in the mitigation step

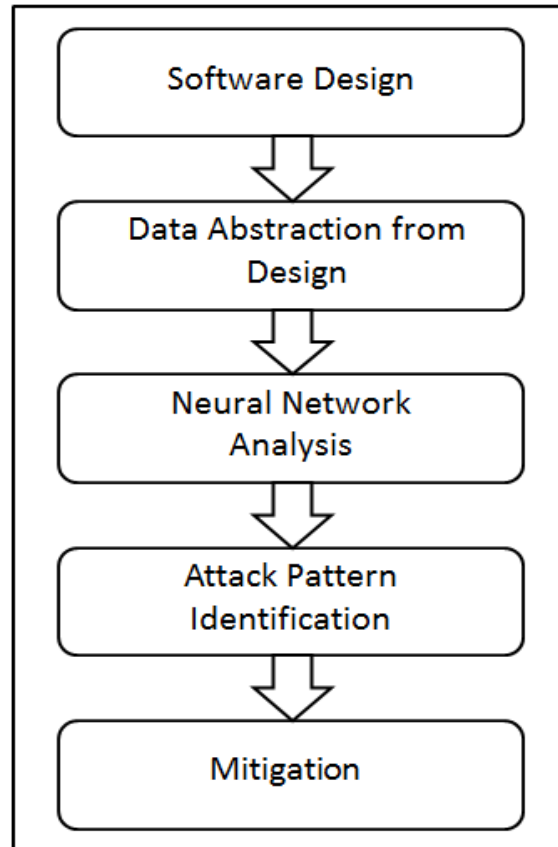


Figure 3. 5: Neural Network 1 Evaluation process steps

3.4.2 Neural Network II

In the second neural network in the neural network module of the model overview, this research aims to use neural network to suggest possible solutions to the attack patterns identified by the first neural network. This approach builds on the proposed selection criterion by Wiesauer and Sametinger (2009) which matches attack patterns to security design patterns. To achieve this, data was abstracted from the 51 regularly expressed attack patterns by William and Gegick (2006). Also, using Microsoft threat classification scheme (STRIDE) the attack patterns were grouped into six groups in order to align the attack patterns to their corresponding threat category. The data abstracted from the attack patterns formed the attributes of the attack patterns that were used in training the neural network. The attributes consists of:

The Attack ID: This is the unique ID that identifies the attack

Resource Attacked: This is the resource that is attacked in the attack pattern.

Attack Vector: This is method through which the attacker uses to attack the resource

Attack Type: This state whether the attack is an attack against confidentiality, integrity or availability

The second neural network is used to match attack patterns to security design patterns defined by Kienzle and Elder (2002), Blakley, et.al (2004) and Steel, et.al, (2005). The security design patterns defined by these authors are stated in the Table 3.1 below

Table 3. 1: List of Security patterns

Security patter by Steel, et.al, (2005)	Security Patterns by Blakley, et.al (2004)	Security patterns by Kienzle and Elder (2003)
Authentication Enforcer	Checkpointed System	Account Lockout
Authorization Enforcer	Standby	Authenticated Session
Intercepting Validator	Comparator- Check Fault – Tolerant System	Client Data Storage
Secure Base Action	Replicated System	Client Input filters
Secure Pipe	Error/ Detection/Correction	Directed Session (M)
Secure Service Proxy		Hidden Implementation (M)
Secure Session Manager	Protected System	Encrypted Storage
Intercepting Web Agent	Policy	Minefield
Secure logger	Authenticator	Network Address Blacklist
	Subject Descriptor	Partitioned Application
Audit Interceptor	Secure Communication	Password Authentication
Container Managed Security	Security Context	Password Propagation
Dynamic Service Management	Security Association	Secure Assertion
Obfuscated Transfer Object	Secure Proxy	Server Sandbox
Policy Delegate		Trusted Proxy
Secure Service Façade		Validated Transaction
Secure Session Object		
		Build Server From Ground up
Message Inspector Gateway		Choose the right stuff
Secure Message Router		Document Security goals
Message Inspector		Document Server Configuration
		Enrol by Validating out of band
Assertion Builder		Enrol Using Third party Validation
Credential Tokenizer		Enrol with pre-existing Shared Secret
Single Sign On (SSO) Delegator		Enrol without validating
Password Synchronizer		Log for Audit

		Patch Proactively
		Red Team the Design
		Share responsibility for security
		Test on a staging server

3.5. The Output Module

In the output module of the model overview, the result of the evaluation of the two neural networks in the neural network module is presented (i.e. the identified attack pattern and the matched security design pattern). Table 3.2 shows the regularly expressed attack pattern proposed by Gegick and Williams (2006) and the security design patterns defined by Steel, et.al, (2005).

Table 3. 2: Security Patterns matched with Attack Patterns

Regular Expression ID	Regular Expression	Description	Security Pattern
Regex1	(User ⁺) (Server ⁺) (Log ⁺) (HardDrive ⁺)	A user can exceedingly access a server that logs accesses to the hard drive. If permitted, the log file may become large enough to fill the hard drive causing the system to crash -- a denial- of-service attack (DoS). This may also occur on servers that log errors.	Authorization Enforcer
Regex2	(User) (Message) (Server) (Header ⁺) (MessageHeaderHandler) (Memory + CPU)	A user may send a message with thousands of headers (e.g. MIME headers) to a server, causing a server memory/CPU DoS.	Intercepting Validator
Regex3	(User) (HTTPServer) (GetMethod) (GetMethodBufferWrite) (Buffer)	A user that submits an excessively long HTTP GET request to a web server may cause a buffer overflow. Either the requestURI or HTTP version may be too long for the buffer. The attacker may be able to escalate their privileges.	Intercepting Validator Message Inspector

Regex4	(User) (Variable + Filename + Header) (HTTPServer) (PostMethod) (BufferWrite) (Buffer)	A user that submits an excessively long POST request via a variable, Filename or Header, may cause a buffer overflow on the server. The POST request may be in the form of a hidden variable, filename or header). The attacker may be able to escalate their privileges.	Intercepting Validator Message Inspector
Regex5	(User) (Server) (Message) (HeaderFieldBufferWrite) (Buffer)	A user may submit an excessively long header field value causing a buffer overflow on the server (e.g. HTTP, email headers). The attacker may be able to escalate their privileges.	Intercepting Validator Message Inspector
Regex6	(User) (HTTPServer) (HTTPMessageHandler) (Log) (SysAdmin) (LogEntryRead) (BufferWrite) (Buffer)	A user that submits an excessively long message to the server can later induce a buffer overflow when viewed by a system administrator. It is possible for the attacker to escalate their privileges.	Intercepting Validator Secure Logger
Regex7	(User) (HTTPServer) (PostMethod) (HTTPContentLengthHeaderValue) (HTTPMessagePayloadLength) (ServerConnectionState)	A user may submit a value via the POST method that specifies the Content-Length of the HTTP header be less than the content-length of the message, thus causing the socket to stay open (DoS). (see regex37)	Intercepting Validator Message Inspector
Regex8	(User) (UserNameEntry) (PasswordEntry) (Server) (AuthenticationRoutine) (BufferWrite) (Buffer)	A user that submits an excessively long string of characters for either the username or password may cause a buffer overflow in the authentication routine. The attacker may be able to escalate their privileges.	Intercepting Validator
Regex9	(User) (SQLInput) (Server) (WebApplication) (Database) (Data)	Failure to sanitize user input (e.g. Query string) can allow a user to submit an arbitrary SQL query, thus allowing for unauthorized access to data. This regex is too abstract to cover the many possibilities of invalid SQL input.	Intercepting Validator Message Inspector

Regex10	(User) (SQLInputField) (Server) (WebApplication) (Database) (CPU)	An attacker may submit a malicious SQL query (such as a Cartesian join of all Tables) consuming the CPU.	Intercepting Validator Message Inspector
Regex11	(User) (CommandLineArgumentEntry) (ApplicationServer?) (Application) (CommandLineArgumentBufferWrite) (Buffer)	A user may submit an excessively long command line parameter causing a buffer overflow. The attacker may be able to escalate their privileges.	Intercepting Validator Message Inspector
Regex12	(User ⁺) (HTMLPage ⁺) (Server ⁺) (HardDrive ⁺)	A user may submit an excessive amount of data in an HTML page, thus filling up the hard drive on which the server resides.	Message Interceptor
Regex13	(User) (InjectionOfMaliciousHTMLTags/scriptInURL/Form) (Cookie*) (FormData*) (ServerVariables*) (Information)	A user may inject malicious scripts/tags (SCRIPT, OBJECT, APPLET, EMBED, FORM) or variables (e.g. JSP, ASP, search string) in a web page, msg. board, email, message (e.g. IM), Script in URL, URL parameter or HTML/CSS TAG, or HTML injection in HTML tag to obtain access to information such as cookies. This is called Cross Site Scripting (XSS).	Intercepting Validator
Regex14	(User) (Machine) (SyslogFunction) (Log) (Memory)	It is possible to corrupt memory by passing format strings through the Syslog(), a logging function. This may potentially be exploited to overwrite arbitrary locations in memory with attacker- specified values. The Syslog function is often improperly used and is thus a target of attacks. Machine is any computer that uses the syslog function.	Intercepting Validator

Regex15	(User) (ReadUserInput) (EnvironmentVariableWrite) (Buffer)	A user may submit an excessively long environment variable causing a buffer overflow in the application. The attacker may be able to escalate their privileges.	Intercepting Validator
Regex16	(User) (GUI/Browser) (BookMarkSave) (BookmarkBufferWrite) (Buffer)	A user may save an excessively long bookmark and cause a buffer overflow. The bookmark may be written by the attacker or come from a long web page title. The attacker may be able to escalate their privileges.	Intercepting Validator
Regex17	(User) (Application) (File) (FileRead)	An application that reads a file may throw an exception or halt if the file is corrupt or has been tampered with by an attacker.	Intercepting Validator
Regex18	(SocketRead) (SocketBufferWrite) (Buffer)	A user may submit an excessively long stream to a socket and cause a buffer overflow. This is true for handling any connection on the internet (e.g. GET request). The attacker may be able to escalate their privileges.	

Rgex19	(Class) (Subclass) (OverriddenSecuredMethods) (Application)	Overriding methods that have been secured in a super class may create a software vulnerability. In Netscape 4.0 the ClassLoader overrode the definition of built-in "system" types like java.lang.Class - applications usually subclass ClassLoader - a better example is from http://java.sun.com/j2se/1.4.2/docs/guide/security/jce/JCERefGuide.html - suggests to not override methods/constructors in CipherInputStream because the class takes into account many security considerations.	
Regex20	(User) (Hyperlink) (Server) (HyperlinkBufferWrite) (Buffer)	A user may make an excessively long hyperlink on a webpage and cause a buffer overflow on a server. If the hyperlink is used to connect to a session, then the malicious user may take over the application.	Message Inspector
Regex21	(User) (Server) (MessageHeaderHandler) (Server)	A user may send a negative, NULL, or invalid value (e.g. not include ":" between header name/value) in a header field resulting in a DoS on the server.	Message Inspector
Regex22	(UserInput) (PointerDereference) (Application)	A user may fail to submit a username causing a DoS. This could be the result of a pointer that is dereferenced to obtain the username, but NULL is returned instead.	Message Inspector

Regex23	(User ⁺) (Server ⁺) (CPU ⁺) (HardDrive*)	A script that make an excessive number of connections to the listening daemon process of a server may cause a DoS. This script need only make connections -- further I/O may not be necessary with the connections.	Message Interceptor Gateway
Regex24	(UserInput) (IntegerEvaluationRoutine) (BufferWrite) (Buffer)	A user that supplies an integer larger than the integer variable type expected may cause an exception/buffer overflow or DoS.	Intercepting Validator
Regex25	(User) (HTTPServer) (GetRequestRoutine) (Application + Information)	A malformed URL (e.g. excessive forward slashes, directory traversals, special chars such as "*", Unicode chars, format string specifier, NULL) may cause a DoS or in case of directory traversal the user may obtain private information.	Intercepting Validator Authorization Enforcer
Regex26	(User) (Server) (SearchString) (Information)	A user may insert a directory traversal such as "../.." in a search string (e.g. CGI) and obtain private information.	Intercepting Validator
Regex27	(User) (SearchString) (Server) (Data) (User) (BufferWrite) (Buffer)	A user that requests data from an untrusted server may receive large data and result in a buffer overflow. Often happens in gaming environments.	Intercepting Validator
Regex28	(Read) (FileHeader) (BufferWrite) (Buffer)	A user may label a file with an excessively long filename and cause a buffer overflow in the process reading the file. This occurred in an operating system context.	Message Inspector

Regex29	(User) (EmailHeader) (Firewall) (Buffer)	A user can overflow a buffer in their firewall with a large email header to escalate their privileges (the user can attack their own company's LAN).	Message Inspector
Regex30	(User) (MalformedDTD) (SOAPServer) (XMLParser) (CPU + Memory)	A user that submits a malformed DTD may cause the XML parser of a SOAP server to consume the CPU/Memory.	Message Inspector
Regex31	(User) (HTTPRequest) (ProxyServer) (BufferWrite) (Buffer)	A user that submits an excessively long HTTP GET request to a proxy server may cause a buffer overflow. The attacker may be able to escalate their privileges.	Message Inspector Gateway
Regex32	(User) (RequestMessage) (Router)	A user that submits malformed headers (e.g. failing to supply expected headers) may cause a DoS. Also, NULL as a header value may cause a DoS.	Message Inspector
Regex33	(User) (HTTPgetRequest) (Router) (EmbeddedServer) (Bufer*)	A user that sends an excessively long GET request to a router may cause a DoS via a buffer overflow or CPU consumption.	Intercepting Validator
Regex34	(User+) (HTTPServer+) (GetRequestRoutine+) (Buffer + CPU)	A user may submit consecutive multiple long GET request URIs to either consume the CPU or overflow a buffer.	Message Inspector Gateway
Regex35	(User) (HTTPgetRequest) (Router)	A user may submit a malformed GET request (e.g. a blank (NULL)) request and cause a router to DoS.	Intercepting Validator

Regex36	(User) ((FTPCommand+MailCommand)+ OSCommand) (FTPServer + MailServer)) (BufferWrite) (Buffer)	A user that submits an overly long OS command or FTP/Mail command may cause a buffer overflow in the FTP/Mail server. The attacker may be able to escalate their privileges.	Intercepting Validator
Regex37	(User) (Socket) (Server) (ExceptionThrown*) (Server)	A user may cause an exception to be thrown in the server and cause it to hang. (No data needs to be transferred) (similar to regex 7)	Secure Pipe Secure Base Action
Regex38	(User) (UserNameEntry) (PasswordEntry) (AuthenticationServer?) (AuthenticationRoutine)	A user that submits a malformed username or password for authentication may cause a DoS (e.g. format string specifier) or NULL as part of the name may bypass the authentication routine.	Authentication Enforcer Intercepting Validator
Regex39	(User) (FTPRequest) (FTPServer) (BufferWrite) (Buffer)	A user may submit a long directory request (e.g. in the URL of a browser) by using long directory names or "/" can cause a buffer overflow or DoS in the FTP server. The attacker may be able to escalate their privileges.	Intercepting Validator
Regex40	(User) (FTPRequest) (FTPServer) (GetRoutine) (Server)	A user that requests a file that does not exist on the server may cause a DoS (e.g. Get <unavailable file>)	Message Inspector Gateway
Regex41	(Metafile) (SizeField) (FileHeader) (FileRead) (BufferWrite) (Buffer)	A user that specifies the "Size" field of a metafile to be less than the actual file may cause a buffer overflow.	Message Inspector
Regex42	(Application) (DownloadMaliciousFile) (PredicTableFileLocation) (AttackerReference) (Information)	A user that saves files to predicTable locations especially where applications let you reference them may allow for information disclosure.	Obfuscated Transfer Object

Regex43	(Application) (FileCreation) (System)	If an application creates a file/directory that allows malicious users to write to them (makes them symbolic links or simply changes them), then attackers can escalate their privileges.	Obfuscated Transfer Object
Regex44	(ApplicationRun) (Privileges) (System)	An application that runs with SYSTEM privileges and lets a user execute another program such as CMD.EXE may grant themselves SYSTEM privileges.	Container Manager Security
Regex45	(User) (MessageHeader+QueryParam)) (Server) (System)	A user may insert shell commands into a message handler on a server (e.g. email server), which may allow the attacker run those commands on that system.	Message Inspector
Regex46	(User) (Message) (Server) (System)	A user that submits a message (command) to the server before authentication may cause a DoS (done in C code).	Authentication Enforcer Message Inspector Gateway
Regex47	(SourceFile) (IncludeFile) (EnvironmentVariable+ProgramVariable+URLparam) (System)	An attacker can change/influence an environment, program, or URL variable to point to a remote machine. If the variable points to an "include" directory, then the attacker's include file can be executed on the target system	Intercepting Validator
Regex48	(User) (MalformedFTPCommand) (FTPServer) (BufferWrite) (Buffer)	A user that submits an excessively long FTP command may cause a DoS or buffer overflow.	Intercepting Validator

Regex49	(User) (InvalidRequest) (ErrorMessage) (System)	A user that submits an invalid request may be returned with an error message that shows the installation path of the server.	Obfuscated Transfer Object Intercepting Validator
Regex50	(User) (Application) (Subprocess) (System)	An application that spawns a sub process to handle a user command must ensure that the sub process does not have elevated permissions.	Container Manager Security
Regex51	(WebBrowser) (CLSID) (Filename) (System)	A user that embeds a CLSID in the filename of a malicious file can trick a web browser into opening the file with a different application than intended.	Authentication Enforcer Message Inspector Gateway
Regex52	(Server) (QueryString) (Command) (System)	A user may insert a command for the value of a URL parameter and execute that command on the server (remote execution attack)	Intercepting Validator
Regex53	(User) (URL) (Server) (Device) (System)	A user that submits a URL with a device as part of the request may cause a DoS (e.g. http://[victim]/COM1)	Intercepting Validator

3.6. Summary of Chapter three.

When potential attack patterns are matched against software design, the software developers are able to take the necessary steps in mitigating the security flaw identified in the design. To achieve this, the proposed neural network tool in research work matches the security flaws in the design to possible attack patterns. The data source and attributes used in abstracting the information from the attack scenarios has been highlighted in this chapter. To suggest possible solutions to the flaws identified with the attack pattern, the second neural network matches the attack patterns to security design pattern that can provide mitigation. The attributes for abstracting the information used in training the second neural network and the security design patterns which are matched to attack patterns were highlighted. Finally, the result of the output module of the model overview was discussed. In the next chapter the implementation of the first neural network will be demonstrated.

Chapter 4. Implementation of Neural Network I

4.1. Introduction

This chapter demonstrates the implementation of the first neural network in the neural network module discussed in chapter three. This include demonstration on how data is abstracted from attack scenarios reported in authoritative data sources using the attack attributes for training the neural network; demonstration on how the training data is encoded, the implementation of the neural network architecture and the training of the neural network. The analysis of performance of the network is also presented in this chapter.

4.2. Data Collection

A total of 715 attacks from the online vulnerability databases (see chapter 3) relating to 52 of Gegick and Williams' regular expressed attack patterns were analysed. Table 4.1 is sample of data collected on the attacks from the vulnerability databases. At the time of collecting data from the online vulnerability databases, there was not enough data to fully analyse regularly expressed attack pattern 19. However, from the 715 attacks that were analysed, 260 of the attacks were unique in terms of their impact, mode of attack, software component and actors involved in the attack. The remaining 455 attack are a repetition of the same type of exploit in different applications that has been reported in the vulnerability databases (See Figure 4.1). As this research is focused on evaluating software design for security flaws, only attacks that exploited software design flaws were considered in the online vulnerability database. Table 4.1 shows a sample of data collected from the vulnerability databases. A complete list of data collected can be seen in Appendix VII. Once the attack has been analysed the attack attributes discussed in chapter three are used to abstract the data capturing the attack scenario in the exploit for training the neural network.

Table 4.1 Sample of data collected from vulnerability databases

Date published	Title of Vulnerability from Vulnerability Databases	ID
2006-07-12	Shopping Cart Multiple HTML Injection Vulnerabilities	CVE-2006-3542
2007-02-26	Shop Kit Plus StyleCSS.PHP Local File Include Vulnerability	CVE-2007-1127
2009-01- 27	Shop-inet 'show_cat2.php' SQL Injection Vulnerability	CVE-2009-0292
2000-02-01	Multiple Vendor Web Shopping Cart Hidden Form Field Vulnerability	BID -1237
2010-09-02	Shop a la Cart Products Multiple Input Validation Vulnerabilities	BID-42953
2008-01-07	Shop-Script 'index.php' Local Information Disclosure Vulnerability	BID- 27165
2006-10-23	Shop-Script Multiple HTTP Response Splitting Vulnerabilities	BID-20685
2005-05-16	Shop-Script CategoryID SQL Injection Vulnerability	BID-13633
2005-05-16	Shop-Script ProductID SQL Injection Vulnerability	BID-13635

2008-06-08	Shop-Script Pro 'current_currency' Parameter SQL Injection Vulnerability	BID-35429
2011-10-20	Wizmall Multiple Remote File Disclosure Vulnerabilities	BID-50300
2011-10-20	Wizmall Multiple SQL Injection Vulnerabilities	BID-50302
2004-02-16	ShopCartCGI Remote File Disclosure Vulnerability	CVE-2004-0293
2005-12-23	ShopCentrik ShopEngine EXPS Parameter Cross-Site Scripting Vulnerability	BID-16054
2010-05-22	ECShop 'search.php' SQL Injection Vulnerability	BID-40338
2010-05-07	ECShop 'category.php' SQL Injection Vulnerability	BID-40001
2009-05-29	ShopEx ECShop 'integrate.php' Multiple Remote Command Execution Vulnerabilities	BID-44497
2009-04-27	ECShop 'user.php' SQL Injection Vulnerability	BID-34733
2010-02-06	ShopEx Single 'errinfo' Parameter Cross Site Scripting Vulnerability	BID-39941
2009-08-21	Shopmaker Local File Include and SQL Injection Vulnerabilities	BID-35937
2008-10-22	ShopMaker 'product.php' SQL Injection Vulnerability	BID-31854
2008-02-01	CandyPress Multiple Input Validation Vulnerabilities	CVE-2008-0546
2007-10-23	CandyPress Store Logon.ASP Cross-Site Scripting Vulnerability	CVE-2007-5629
2007-01-09	Shopstorenow E-commerce Shopping Cart Orange.ASP SQL Injection Vulnerability	CVE-2007-0142
2010-04-06	ShopSystem 'view_image.php' SQL Injection Vulnerability	BID-39260
2006-11-15	ShopSystems Index.PHP SQL Injection Vulnerability	CVE-2006-5935
2006-04-11	ShopWeezle Multiple SQL Injection Vulnerabilities	CVE-2006-1706
2005-08-01	Opera Web Browser Download Dialog Manipulation File Execution Vulnerability	CVE-2005-2407
2012-02-07	WordPress AllWebMenus Plugin 'actions.php' Arbitrary File Upload Vulnerability	CVE-2012-1010
2010-07-02	Qt Remote Denial of Service Vulnerability	CVE-2010-2621
2010-06-16	SolarWinds TFTP Server Write Request Denial Of Service Vulnerability	CVE-2010-2310
2010-06-15	Dlink Di-604 IP Textfield Size Cross-Site Scripting and Denial of Service Vulnerabilities	CVE-2010-2293
2010-06-15	uniper Networks IVE OS 'homepage.cgi' URI Redirection Vulnerability	CVE-2010-2289
2001-09-08	Hassan Consulting Shopping Cart Arbitrary Command Execution Vulnerability	CVE-2001-0985
2004-12-31	Virtual Programming VP-ASP Shopping Cart CatalogID SQL Injection Vulnerability	CVE-2004-2412
2004-11-23	RobotFTP Server Username Buffer Overflow Vulnerability	CVE 2004-0286

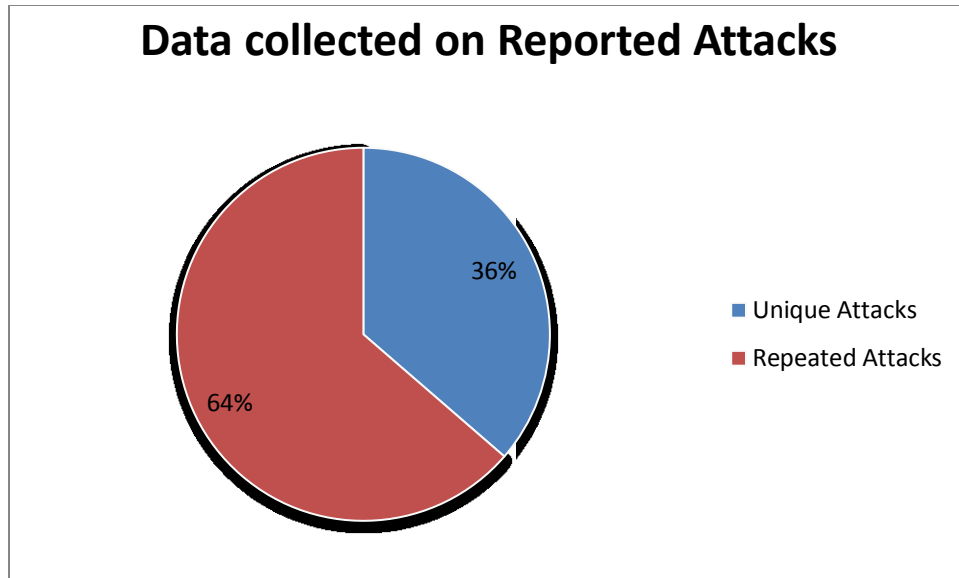


Figure 4. 1: Pie chart of data collected

4.3. Data Encoding

In order to encode the data needed for training the neural network, data collected on the reported attacks from the vulnerability database was analysed. All the attack scenarios analysed from the vulnerability databases exploited software design flaws in the applications they were found. In most cases, the attacks are carried out in ways the software designer did not intend the software application to be used at the time of design. For instance a system that is designed to accept user input can be exploited by a malicious user who enters a long string of characters to cause a buffer overflow. Therefore, all the attack scenarios analysed were actual software design scenarios containing flaws that were exploited by attackers. Information provided on each reported attack was used in modelling the attack scenarios in the attacks. Figure 3.2 in chapter 3 is an example of attack on webmail in the CVE detail online vulnerability database. This attack corresponds to regularly expressed attack pattern 3 (See Table 3.2 in Chapter 3, Section 5). This attack pattern is:

(User)(HTTPServer) (GetMethod) (GetMethodBufferWrite) (Buffer)

In order to get a clearer picture of the attack, the attack modelling concept of the secure troopos was used to analyse the attacks (See chapter for 2 for further discussion). Using this approach, the attack components such as the attacker, the actors and resources under attack and their interaction were clearly identified in the design. Figure 4.1 below shows how the attack on webmail was analysed using this approach.

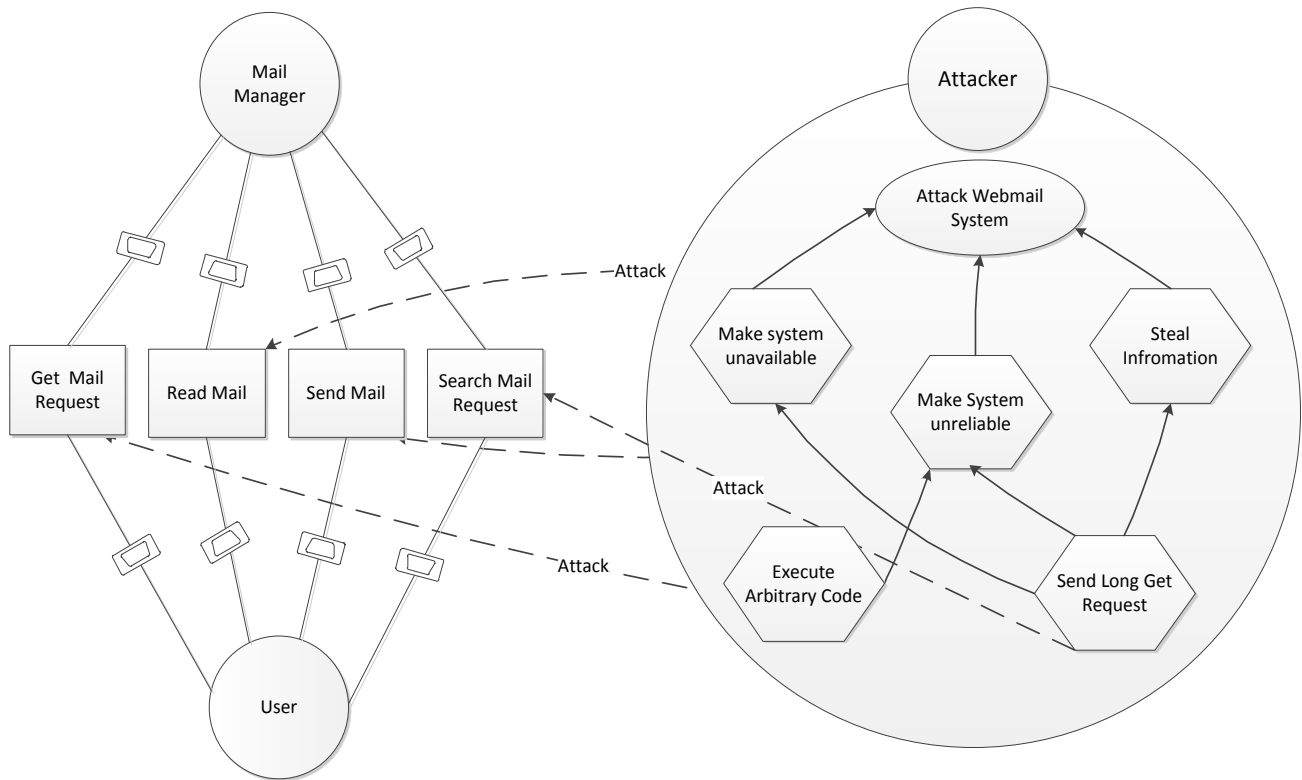


Figure 4. 2: Analysis of Attack on Webmail using secure tropos approach

From the figure above, it could be seen that the goal of the attacker is to attack the webmail system. To achieve this, he can perform any of the following tasks:

- **Make system unavailable:** The attacker can accomplish this task by sending long Get Request to manipulate the URL when playing the role of the user. The Mail Manager (*dependor*) depends on the User (*dependee*) for Get Mail Request and Send Mail Request (i.e. the *dependun*) which are resources the attacker can manipulate when performing this task.
- **Make system unreliable:** The attacker can also execute arbitrary code as a sub task to be performed in order to make the system unreliable. This task can be performed in a multi-stage attack where the attacker has initially escalated his privileges by performing the task above followed by running arbitrary codes through the Search Mail or Get Mail Request resource
- **Steal Information:** The attacker can perform this task by attacking Read Mail and Send Mail resources. The attacker can also escalate his privilege using the sub-task (i.e. send Long Get Request) on Get Mail Request and Send Mail Request resources to cause a buffer overflow. An attacker gaining the privileges of the administrator will be able to access the information on any user account on the webmail server.

Once the data collected on the attacks from the vulnerability databases has been analysed, the attack scenario in the attack is established. Using the attack attributes in Table 4.1, the data needed for training the neural network is abstracted from the attack scenario.

Apart from using the secure troopos approach to analyse the data collected on the attacks in the vulnerability database, the sequence diagram was also used. Figure 4.2 gives two possible design scenarios when analysing the webmail attack in figure 3.2 in chapter 3. This involves the client who could be the attacker requesting for access from the webserver by presenting his login information. In response, the webserver retrieves the client details from the database, validates the client and gives a notification that the login is successful. The client then proceeds to request for mail access. The webserver redirects the request to the mail server who in turn retrieves the mail from the database for the webserver. The webserver then sends the mail to the client. It should be noted that webserver, login information, mail server and database in the sequence diagram are resources which the attacker may be choosing as targets for his attacks

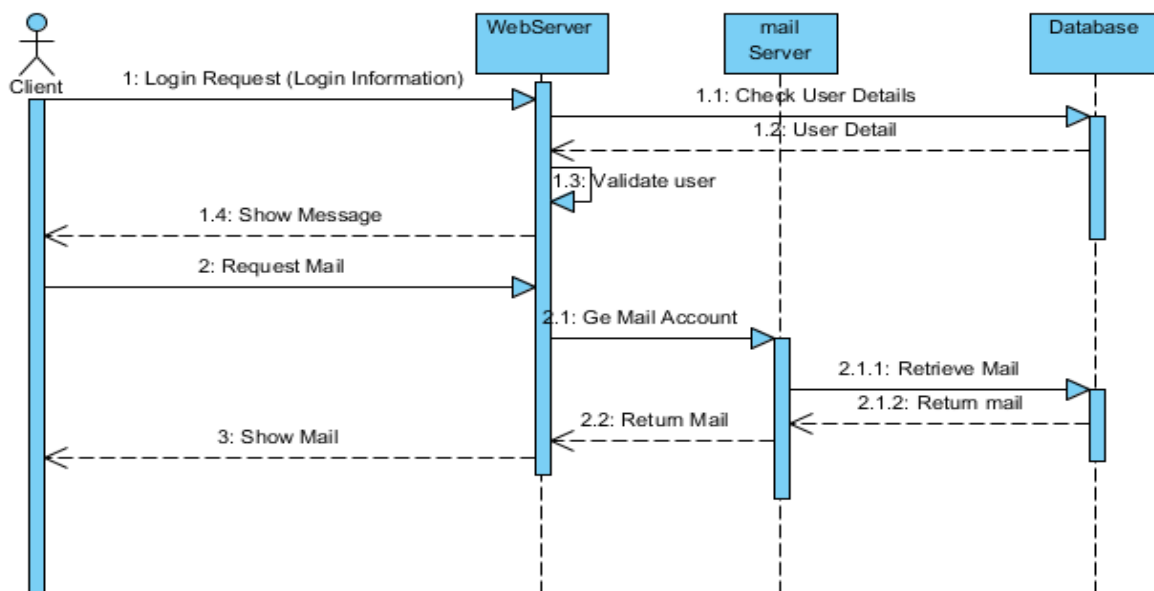


Figure 4. 3: Sequence diagram on Webmail

From the sequence diagram, two design scenarios i.e. the login scenario and the request for mail scenario were established. Using the Long Get Request, during the two design scenarios, the attacker can stage a buffer overflow attack. Through this way, attack scenarios were also established for the purpose generating data for training the neural network.

The training data sample consists of 12 input units for the neural network. This corresponds to the number of the attributes used in abstracting data from the attack scenarios. Table 4.1

shows gives an overview of the attack attributes and the code for each data abstracted using the attribute.

Table 4. 1: List of Attack Attributes

S\N	Attributes	Observable	Value
1	Attacker	No access	0
		Read access	1
		Change Access	2
		Delete All	3
2	Source of Attack	External	1
		Internal	2
3	Target of Attack	Data resource	1
		System resource	2
4	Attack Vector	Software component used for attack	Component ID
5	Attack Type	Availability	1
		Integrity	2
		Privacy	3
6	Input Validation	No Validation	3
		Partial validation of inputs	2
		All inputs validated	1
7	Dependencies	None	0
		Authentication	3
		Access Control	3
		Input validation	3
		Trust Boundary undefined	3
		Encryption	3
8	Output encoding to external applications/Services	None	0
		Parameterized Interface	3
		Stored procedure	3
		Escaping all user supplied input	3
9	Authentication	None	0
		Plain text password	5
		Harsh password	25
		Salted password	2
		One time password	3
		Lock outs	3

		Session management	3
		Time outs	3
		Decisions logged	3
		Credential management	3
		Re-authentication for accessing sensitive data	3
10	Access Control	None	0
		Data access authorization	1
		URL access authorization	2
		Service access authorization	3
		Function access authorization	4
		File access authorization	5
		All access path checked	6
		Server side enforcement	10
11	HTTP Security	None	0
		Redirect data validation	3
		Defined HTTP request methods	3
		Input Validation for HTTP request and response	3
		Secure flag set up for cookies with sensitive data	3
		Strong random token	3
12	Error handling and Logging	Sensitive application data in error message	3
		Server side controlled logging	3
		Server side error handled on server	3
		Access not denied by default	3
		Success and failure security relevant event logged	3
		No Access control to log	3
		No Validation of un-trusted data used in event logged	3

To generate the corresponding values for attack attributes four and five in Table 4.1 above, 106 attack components were abstracted from the regularly expressed attack pattern in Table 3.2 in chapter 3 (See Table 4.2 and the appendix for the description on the attack components). Also, to make it easier for the attack components to be identified using the attack attributes above, each attack component was allocated a unique ID and were also classified into the following groups:

- **Users:** This group normally depicts users of the software application system. However, in the context of this research work, the group mostly refer to the attackers or the victims of the attacks

- **Inputs:** This signifies the input the user is supplying to the software application to be processed. An example of this is a search string submitted to a search engine on a web site
- **Data:** Attack components in the regularly expressed attack pattern such as file, information or cookie referring to any for data which is involved in the attack was classified under this group.
- **Parameter:** This group is used classify the attack components used as parameters in software applications. For example, parameter used in establishing the size of a field, memory reference point or the payload length of an HTTP message.
- **Hardware:** This group refer to any hardware such as CPU, hard drive or memory that is involved in an attack.
- **Software:** This group also refer to software applications involved in attacks such as when an attacker takes over a software application that has system privileges to run malicious code. Attack components such as application, browser, firewall or sub process of an application are classified under this group
- **Server:** Attack components representing any server processes in the regularly expressed attack pattern is classified under this group. Example of this include application server, proxy server or FTP server
- **Event:** All the attack components representing system events such as when reading or writing to a file or buffers is classified under this group. Example of this attack component includes buffer write, file read, HTTP Request and Get Method
- **Others:** Three attack components that could not be categorized under the groups above were included in this group. The attack component includes: class, subclass and privilege.

To further analyse the attack components in the regularly expressed attack pattern a concept map in figure 4.3 was used to illustrate how they interact with one another using the groups in which the attack components has been categorized into. Using this concept the following interactions were observed between the attack components.

1. The user or client group interacts with other groups by:
 - **Using *Inputs* to trigger *Events*** and receiving **responses** from *Events*
 - **Having** privileges. It should be noted here that in many attack scenarios the attacker may not need to have any privilege to stage an attack
2. The Event group interacts with other groups by:
 - **Responding** to actions **triggered** by the *Users* or other *Software* systems. E.g. A software application responding to user requests to open a file.
 - **Accessing *Data***. E.g. User triggering the Get Request event to access web pages

3. The Software and Server group comes under the Application/System group in the concept map. They interact with other groups by:

Using *Inputs* to **trigger** *Events* and receiving **responses** from *Events*

- **Having** privileges.
- **Uses** Hardware / Resources

4. The Hardware/ Resource group interact with other groups by

- **Saving, retrieving** or **processing** data when it is **used by** *software* systems
- **Has** privileges

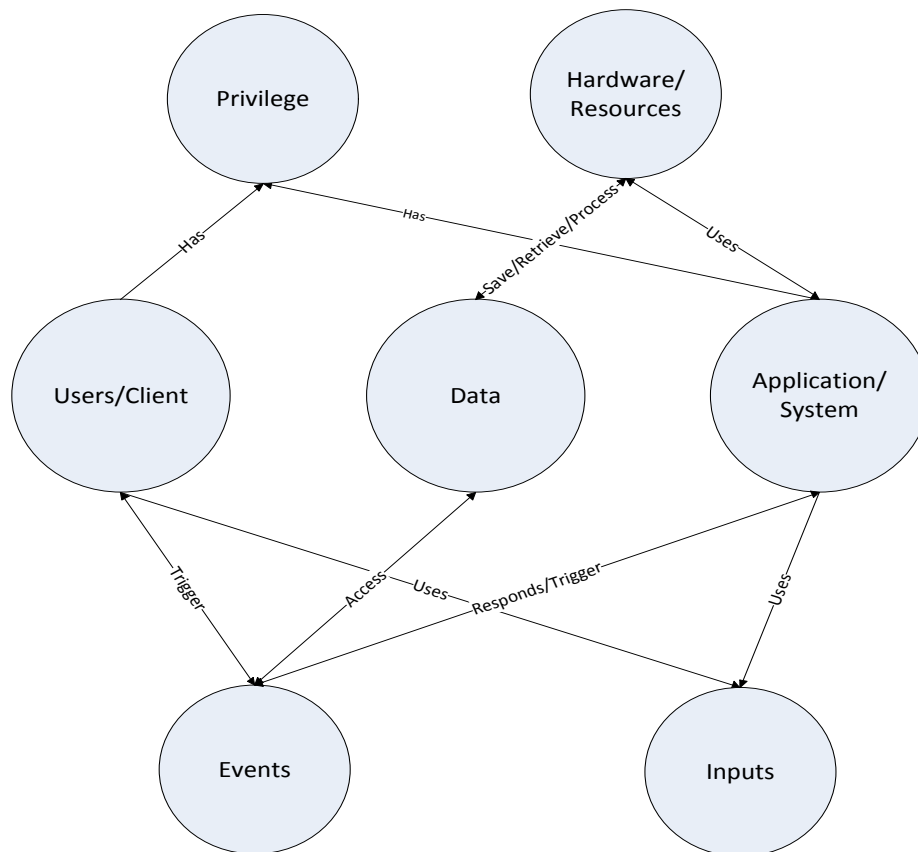


Figure 4. 4: Concept map showing interaction between the groups in which the attack components has been categorized

Table 4. 2: List of Attack Components

Component ID	Software Component	Group
1	Application	Software
2	Application Run	Events
3	Application Server	Parameter
4	Attacker Reference	Parameter

5	Authentication Routine	Event
6	Bookmark Buffer Write	Event
7	Bookmark Save	Event
8	Browser/GUI	Software
9	Buffer	Hardware
10	Buffer Write	Event
11	Class	Other
12	Client	User
13	CLSID	Parameter
14	Command	Inputs
15	Command Line Argument Buffer Write	Event
16	Command Line Argument Entry	Inputs
17	Computer	Hardware
18	Cookie	Data
19	CPU	Hardware
20	Data	Data
21	Database	Data
22	Daemon Process	Software
23	Device	Hardware
24	Download Malicious File	Data
25	Email Header	Data
26	Embedded Server	Server
27	Environment Variable	Parameter
28	Environment Variable Write	Event
29	Error Message	Data
30	Exception Thrown	Event
31	File Creation	Event
32	File Header	Data
33	Filename	Data
34	File Read	Event
35	Firewall	Software
36	Form Data	Data
37	FTP Request	Event
38	FTP Server	Server
39	Get method	Event
40	Get Request Routine	Event
41	Get Routine	Event
42	Hard Drive	Hardware
43	Header	Data
44	Header field Buffer Write	Event
45	HTML Page	Data
46	HTTP Content	Data

47	HTTP Get Request	Event
48	HTTP Message Handler	Event
49	HTTP Message Payload length	Parameter
50	HTTP Request	Event
51	HTTP Server	Server
52	Hyperlink	Parameter
53	Include File	Data
54	Information	Data
55	Integer Evaluation Routine	Event
56	Invalid Request	Inputs
57	Length Header Value	Parameter
58	Log	Data
59	Log Entry Read	Event
60	Malformed DTD	Data
61	Malformed FTP Command	Inputs
62	Malicious client	User
63	Malicious Include File	Data
64	Memory	Hardware
65	Message Header	Data
66	Message header Handler	Event
67	Metafile	Data
68	Overridden Secured Methods	Event
69	Password Entry	Input
70	Pointer Dereference	Event
71	Post Method	Event
72	PredicTable File Location	Parameter
73	Privileges	Other
74	Program Variable	Parameter
75	Proxy Server	Server
76	Query Parameter	Parameter
77	Query String	Input
78	Read User Input	Event
79	Request Message	Event
80	Search String	Input
81	Server	Server
82	Server Connection State	Event
83	Server Variables	Parameter
84	Size Field	Parameter
85	SOAP Server	Server
86	Socket	Software
87	Socket Buffer Write	Event
88	Socket Read	Event

89	Source File	Data
90	SQL Input	Input
91	SQL Input Field	Input
92	Subclass	other
93	Sub process	Software
94	Sys admin	User
95	Syslog Function	Event
96	System	Software
97	URL	Input
98	URL param	Parameter
99	User	User
100	User Input	Input
101	Username Entry	Input
102	Variable	Parameter
103	Victim Client	User
104	Web App	Software
105	XML Parser	Software
106	FTP Handler	Event

To demonstrate how the data for training is encoded using the code for each of the attributes in the Table above, information was abstracted on the attack scenario on webmail by looking at the online vulnerability databases to get the details of the attributes we are interested in (See chapter 3). In this example, the data abstracted from the attack scenario using the attack attribute list is as follows.

Table 4. 3: Sample of Pre-processed Training Data from Attack Scenario

Attacker	Source	Target	Attack Vector	Attack Type	Input Validation	Dependencies	Output Encoding	Authentication	Access Control	HTTP Security	Error
No Access	External	Buffer	Long Get Request	Availability	Partial Validation	Authentication & Input Validation	None	None	URL Access	Input Validation	None

Using the corresponding values for the attributes, the data is then encoded as shown in Table 4.4 below.

Table 4. 4: Sample of Training data after encoding

Attacker	Source	Target	Attack Vector	Attack Type	Input Validation	Dependencies	Output Encoding	Authentication	Access Control	HTTP Security	Error
0	1	9	39	5	2	6	0	2	2	3	0

The second stage of the data processing involves converting the encoded data into ASCII comma delimited format which can be used to train the neural network as shown below

0, 1, 9, 39, 5, 2, 6, 0, 0, 2, 2, 3, 0

Finally, the data is loaded in the neural network for training as shown in the following Table.

Table 4. 5:Sample of data input in Neural Network

Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Input 10	Input 11	Input 12
0	1	9	39	5	2	6	0	0	2	3	0

The data for the expected output of the neural network are derived from the attack pattern that is to be identified in each of the attack scenarios. Therefore, each of the attack patterns is given a unique ID to derive the expected output for each of the input data samples. The unique ID corresponds to the number of the regularly expressed attack pattern as shown in Table 3.2 in chapter 3. The output data sample consists of output units corresponding to the attack pattern IDs. For example the neural networks implemented in this chapter classify attacks into different attack patterns. The above sample data on webmail attack corresponds to regularly expressed attack pattern 3. Therefore, the neural network is trained to identify the expected attack pattern as 3 using the following output data

0, 0, 3, 0,

4.4. The Neural Network Architecture

The feed-forward back-propagation neural network is used to evaluate scenarios from software designs and identify possible attacks in the design. The back-propagation neural network is a well-known type of neural network commonly used in pattern recognition problems (Srinivasa and Settupalli, 2009). It has been used in this research because of its simplicity and reasonable speed. The three layer back-propagation architecture (see Figure 4.4) was adopted in this research work for training the neural network. The choice of this architecture is mainly due to the fact that it is the most standard and general architecture commonly used for training neural networks (Ryan, Lin and Mikkulainen, 1998). Therefore, by adopting this architecture, the feasibility of the proposed neural network tool in this research can be demonstrated and its results can be replicated easily.

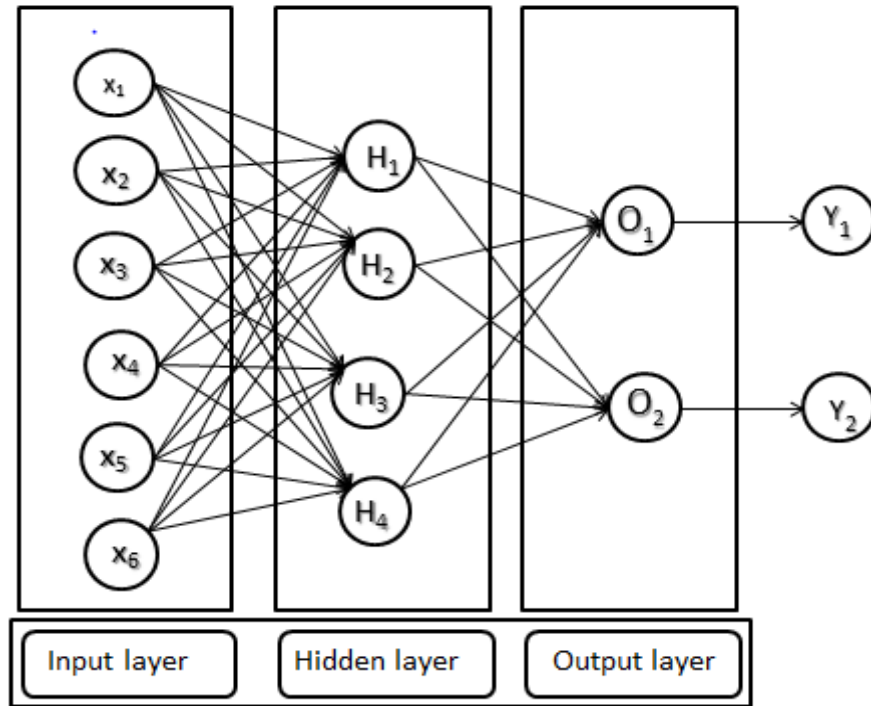


Figure 4. 5: The Neural Network Architecture

To find out the optimal performance of the neural network, three different training optimization algorithms were applied to the back propagation network. The tree training algorithms has been selected because they are commonly used in training neural networks for resolving pattern recognition problems. The training algorithms consisted of the following:

- **Levenberg-Marquardt (LM):** This training optimization algorithm has been described as the fastest training algorithm. It is also the recommended first choice for supervised neural network. The advantage of the LM optimization algorithm is noticeable mostly on function approximation problems with relatively small size neural network where very accurate training is required (Gavin, 2011, Beale, Hagan and Demuth et.al, 2010, Lourakis, 2005). Compared to other training optimization algorithms, LM training algorithm also obtains a lower mean square error (MSE). However LM training algorithm is less efficient when used with large neural network because it requires more memory and computational time. Also, in pattern recognition problems, the performance of LM training optimization algorithm is less efficient (Beale, Hagan and Demuth et.al, 2010). Based on the advantages of LM training algorithms highlighted above, it has been considered for training the neural network in this research work
- **Resilient Back-propagation (RP):** RP optimization training algorithm has been described as the fastest algorithm for training neural networks on pattern recognition problems (Beale,

Hagan and Demuth et.al). It requires relatively small memory but when its error goal is reduced, its performance degrades.

- **Scaled Conjugate Gradient (SCG):** The SCG training optimization algorithm give a good performance on various types of problems especially where the size of the neural network is large. It is commonly used in pattern recognition problems because it converges quickly. It is almost as fast RP training algorithm and requires a relative small memory. In comparison with RP training algorithm, when the error is reduced, its performance does not degrade quickly.

Beale, Hagan and Demuth et.al (2010) states that any neural network whose weight, net input, and transfer functions have derivative functions can be trained using the optimization algorithm discussed above. Once any of the conditions below is met, the training of the neural network stops. Beale, Hagan and Demuth et.al (2010) states that these conditions include:

- Reaching the maximum number of epochs (repetitions) is reached.
- Exceeding the maximum amount of time.
- When performance is minimized to the goal.
- The performance gradient falls below the set minimum gradient
- Validation performance has increased more than maximum fail times since the last time it decreased (when using validation).

For each of the above training optimization algorithms applied to the neural network in this research work, different number of hidden neurons was also used to further optimize its performance. Each layer of the hidden nodes in the neural network architecture apply a tan-sigmoid transfer function to the various connection weights and in the output nodes, the linear transfer function is applied to its weight. As back-propagation neural network is a supervised learning architecture, the training set of data discussed in section 2 was used for its training. From this, the neural network derives its weights and parameters. The weights and parameters are computed by calculating the error between the actual and expected output data of the neural network when the training data is presented to it. The error is then used to modify the weights and parameters to enable the neural network to have better chance of giving a correct output when it is next presented with same input.

4.5. The Neural Network training

To train the neural network the training data set is divided into two sets. The first set of data is the training data sets (260 Samples) that were presented to the neural network during training. The second set (52 Samples) is the data that were used to test the performance of the neural network after it had been trained. At the initial stage of the training, it was discovered that the neural network had too many categories to classify the input data into (i.e. 52 categories)

because the neural network was not able to converge. To overcome the problem, the training data was further divided into two sets. The first set contained 143 samples and the second set contained 117 samples. These were then used for training two neural networks. For each of the neural networks, the training optimization algorithms discussed above were applied and its performance was observed when 80, 90, 100, 110 and 120 hidden neurons were used. Mat lab Neural Network tool box is used to perform the training. The training performance is measured by Mean Squared Error (MSE) and the training stops when the generalization stops improving or when the 1000th iteration is reached. The training parameters also include the learning rate which is set to 0.01 with a goal of 0; maximum fail set to 6, a minimum gradient of 0.000001. The results and analysis of the performance of this neural network is presented in section six.

Table 4. 6: Training and Test data sets

Number of samples	Training Data	Test Data
Data Set 1	143	26
Data set 2	117	26
Total	260	52

4.6. Analysis and Discussion

As highlighted above, different numbers of neurons were applied to the neural network I to further optimize its performance. In order to find out when neural network I had the best performance, the training was executed in five simulations to obtain the average results of its performance. The average results on the training time, MSE and number of epoch were used in analysis of the performance of the neural network because the neural network is initiated with random weights during its training and this gives different results. The results of the performance of the neural network when RP and SCG training optimization algorithms were applied are presented in this section. Also, statistical analysis was carried out to establish the significance of the training optimization algorithms applied to the neural network using statistical tools. The result of the performance of the neural network when LM training optimization algorithm is applied is not presented because the performance of the neural network was poor when the training algorithm is applied. It took an average of 15 minutes to complete its training and the result of its MSE was an average of 4.5 which is very far from the set goal.

4.6.1. Mean Square Error (MSE)

Table 4.7 and Table 4.8 show the result of the performance of first neural network based on MSE using different number of hidden neurons. The performance of neural network when SCG training optimization algorithm was applied was considered to be very good as the neural network was able to reach its set goal for training with the different number of hidden neurons

used. The MSE results obtained were below 0.005 for which the neural network could generate an outcome very close to the expected attack pattern. Figure 4.7 shows the plot of the average MSE result for the different number of neurons applied to the network. From this plot it could be noticed that the MSE dropped sharply to the lowest MSE (0.002148) when 90 neurons were implemented. To test the performance of the neural network, the second data sets in Table 4.6 were used to test the two neural networks implemented under neural network I and the result produced an output as close as possible to the expected output (See Chapter 6 for more discussion)

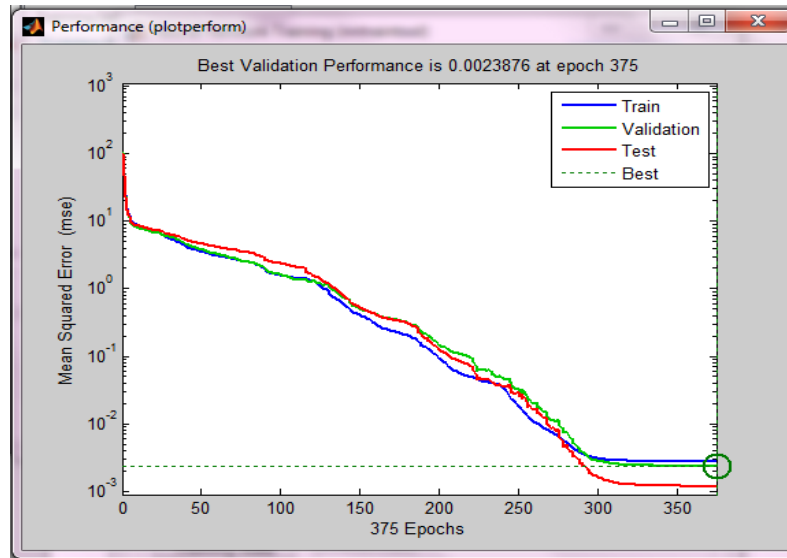


Figure 4. 6: Plot of MSE for Neural Network I during training

Table 4. 7: MSE of Neural Network I with SCG Applied

s\n o	Number of Hidden Neurons				
	80	90	100	110	120
1	0.00279	0.00119	0.00398	0.00199	0.00239
2	0.00199	0.00199	0.000796	0.00239	0.00159
3	0.00447	0.00358	0.002790	0.00279	0.00318
4	0.00389	0.00159	0.001990	0.00279	0.00451
5	0.00309	0.00239	0.00279	0.00199	0.00239
Ave	0.003246	0.002148	0.002469	0.00239	0.002812

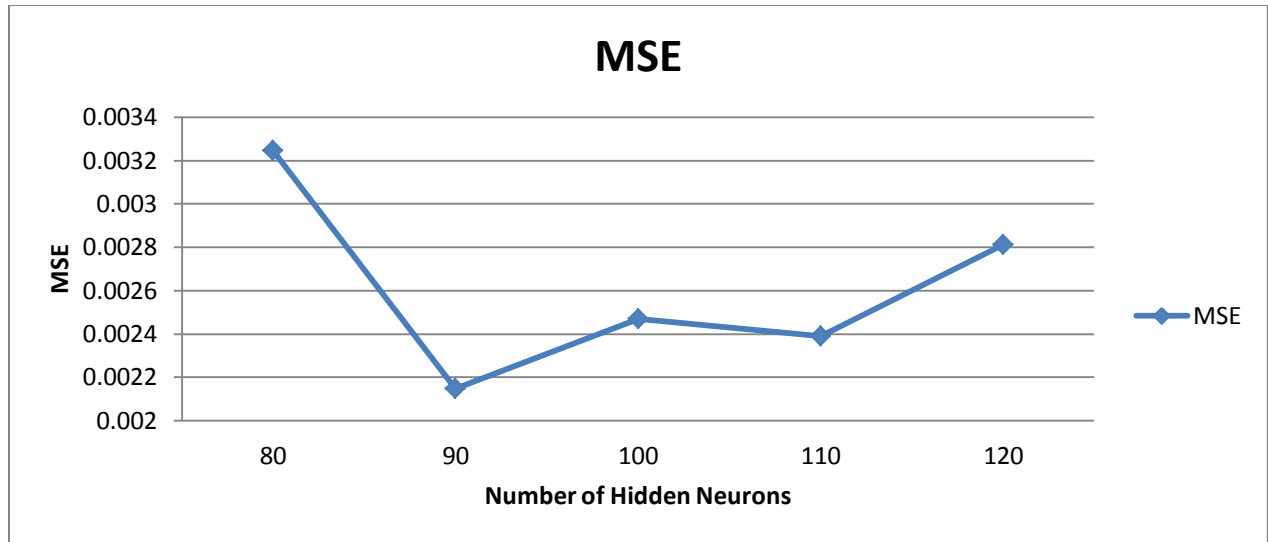


Figure 4. 7: MSE of Neural Network I with SCG Applied

The performance of the first neural network when the RP training optimization algorithm was applied was also considered to be very good. The differences in the average MSE results obtained is small and were also below 0.005. Compared to the MSE result obtained when SCG training optimization algorithm was applied the MSE result obtained for RP was slightly higher. However, when the network was tested with the second data set in Table 4.6 the generated output was also identical to the expected output. Figure 4.8 shows the plot of the average MSE results obtained. The plot showed that the lowest MSE (0.00284) was obtained when 90 neurons were implemented in the neural network.

Table 4. 8: MSE of Neural Network I with RP Applied

s\n o	Number of Hidden Neurons				
	80	90	100	110	120
1	0.00358	0.0019	0.00279	0.00239	0.00239
2	0.00318	0.00402	0.00477	0.00358	0.00279
3	0.00318	0.00159	0.00239	0.0034	0.0042
4	0.00438	0.00404	0.00279	0.00336	0.00407
5	0.00279	0.00265	0.00474	0.00491	0.00239
Ave	0.003422	0.00284	0.003496	0.003528	0.003168

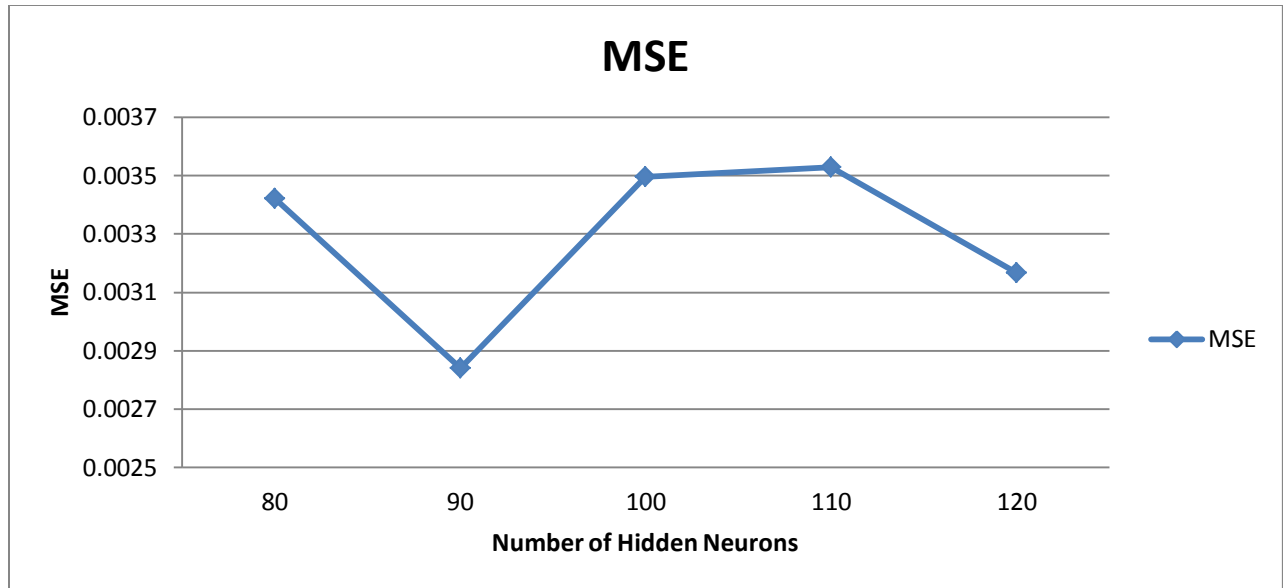


Figure 4. 8: MSE of Neural Network I with RP Applied

4.6.2. Number of Epochs

Table 4.9 and Table 4.10 show the performance of the first neural network based on number of epochs used in training the network. The highest number of average epoch used when SCG was applied as the training optimization algorithm was 462.8 with 100 neurons implemented. This dropped to the lowest (355) when 120 neurons were implemented in the network. Figure 4.9 shows the plot of the average number of epoch used in training with different number of neurons. From the plot, it would noticed that the number epoch increases as the number neuron increased from 80 to 100 and reduces after 110 and 120 neurons were implemented in the network.

Table 4. 9: Number of Epoch used in Neural Network I with SCG Applied

s\n o	Number of Hidden Neurons				
	80	90	100	110	120
1	359	402	585	370	300
2	551	385	379	341	325
3	513	536	514	358	395
4	429	530	424	306	333
5	392	399	412	411	422
Ave	448.8	450.4	462.8	357.2	355

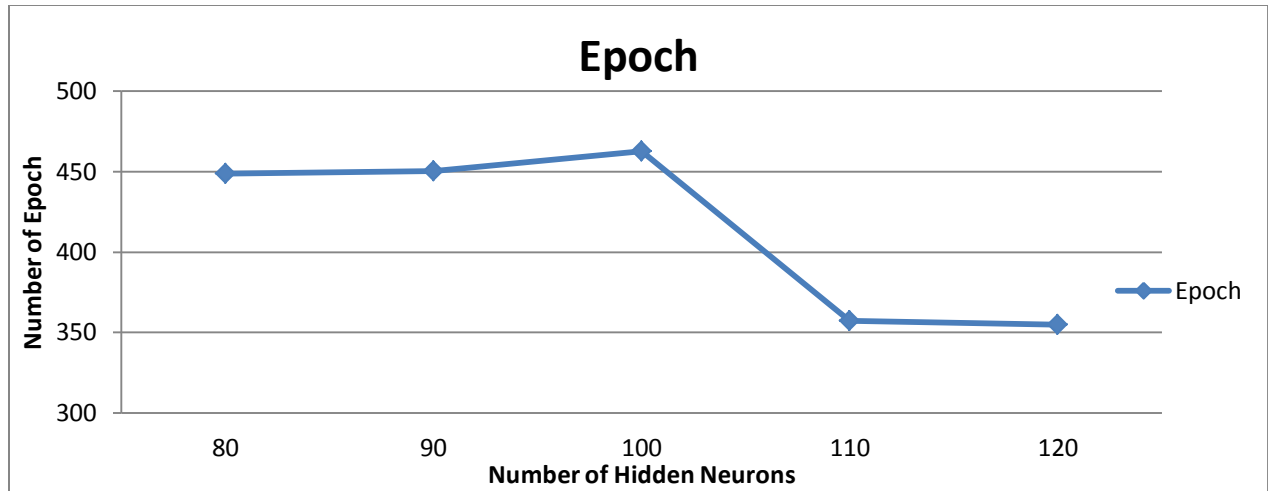


Figure 4. 9: Number of Epoch used in Neural Network I with SCG Applied

The highest number of epoch used in training the network when RP training optimization algorithm was applied is 559.8 when 80 neurons were implemented and the lowest was 454.4 when 100 neurons were implemented. The plot of the average number of epoch used when RP training optimization algorithm was applied in Figure 4.10 shows that the number of epochs used decreases as the number of neurons implemented increases from 80 to 100. However, the number of epoch used increased as number neurons implemented increased from 110 to 120. In comparison to number of epoch used in training the network when SCG training optimization algorithm was applied, the number of epoch used when RP training optimization algorithm was applied was more.

Table 4. 10: Number of Epoch used in Neural Network I with RP Applied

s\n o	Number of Hidden Neurons				
	80	90	100	110	120
1	433	560	551	472	518
2	425	426	425	456	599
3	810	602	483	411	533
4	615	253	395	504	438
5	516	483	418	441	515
Ave	559.8	464.8	454.4	456.8	520.6

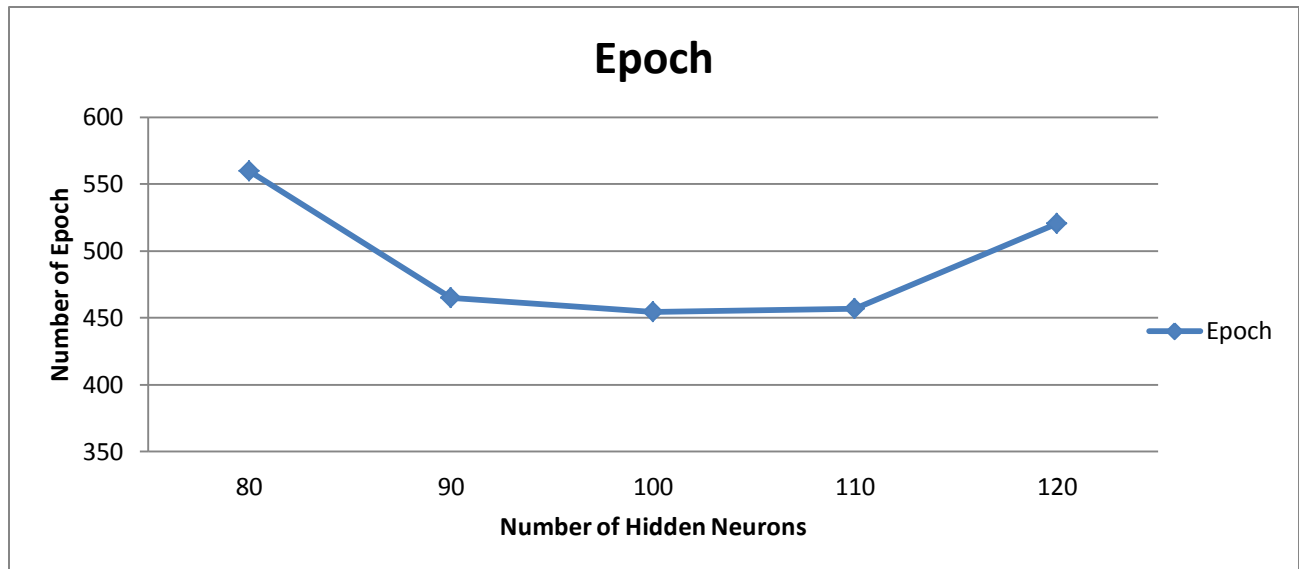


Figure 4. 10: Number of Epoch used in Neural Network I with RP

4.6. Summary of Chapter 4

The implementation of the first neural network has been demonstrated in this chapter. Data from online vulnerability databases were used as the data source for the information needed on the attack scenarios used in training the neural network. To model the attack scenario from the data, secure tropos approach and sequence diagrams were used. This provided all the needed information on the attack components in the attack. Using the 12 attack attribute, data was abstracted from the attack scenarios and then encoded. Further analysis was done on the attack attribute four and five in order to encode the data. This involved generating all the attack components from the regularly expressed attack patterns, giving each attack component a unique ID and classifying them into different groups to illustrate how they interact with one another. A discussion on the concept map showing their interaction was presented. The data abstracted was used subsequently as the input data for the neural network. For the expected output data, unique IDs were allocated to the regularly expressed attack patterns which the neural network is trained to identify based on the information on the attack scenario presented to it. The standard three layer neural network architecture was adopted for this research and to obtain an optimal performance for the neural network, three training optimization algorithms were used along with different number of hidden neurons. The training optimization algorithm includes Levenberg-Marquardt (LM), Resilient Back-propagation (RP) and Scaled Conjugate Gradient (SCG). The strength and weakness of all the training optimization algorithms were highlighted. The performance of neural network during training was measured by Mean Square Error (MSE).

The performance of the neural network were analyzed based on MSE, number of epoch used in training and time of training. The analysis of the overall result of the performance of the

network showed that neural network I gave a better performance when SCG training optimization algorithm was applied. The best MSE performance of network was observed when 90 neurons were implemented and the training was completed in 51 seconds. Based on this result, 90 hidden neurons were chosen for the implementation of neural network I and SCG training optimization algorithm was chosen for its training. In the next chapter, the implementation of neural network II is demonstrated.

Chapter 5. Implementation of Neural Network II

5.1. Introduction

This chapter demonstrates the implementation of neural network II the neural network model. The process through which the neural network matches the security patterns to identified attack pattern by neural network I model is demonstrated. This includes the implementation of the neural network, demonstration on data collection and encoding and training of the neural network. The result and analysis of performance of the network is also presented in this chapter.

5.2. Data Collection

The identified 52 regularly expressed attack patterns by William and Gegick was used as the data source for training the second neural network. To align the regularly expressed attack patterns to the threats that can exploit security flaws they represent, Microsoft threat classification scheme (STRIDE) was used to classify them into six groups according to their corresponding threat category. Table 5.1 shows that out of the 52 regularly expressed attack patterns, 1 of them was classified under spoofing identity attack category, 2 was classified under tamper with data attacks, none was classified under repudiation attacks, 6 was classified under the information disclosure attacks, 21 was classified under the denial of service attacks and 27 was classified under the elevation of privilege attacks. No attack was classified under repudiation attacks because none of the regularly expressed attack patterns demonstrated this type of attack. However, it was assumed that this attack was covered under the elevation of privilege attack because the attacker must have escalated his privileges before been able to cover his tracks in a multi-stage attack scenario.

Table 5. 1: Classification of Attack Pattern

	Attack Category	Attack IDS	Frequency
1	Spoofing	51	0.02%
2	Tampering	9, 47	0.04%
3	Repudiation		0.00%
4	Information Disclosure	13, 24, 25, 26, 42, 49	0.12%
5	Denial of Service	1, 2, 7, 10, 17, 21, 22, 23, 24, 25, 32, 33, 34, 35, 37, 38, 39, 40, 46, 48, 53	0.41%
6	Elevation of Privilege	3, 4, 6, 8, 9, 11, 12, 14, 15, 16, 17, 20, 28, 29, 30, 31, 36, 38, 39, 41, 43, 44, 45, 47, 50, 51, 52	0.53%

Data was also collected from the security design patterns defined by Steel, et.al (2005), Blakley, et.al (2004) and Kienzle and Elder (2003). The security patterns defined by these authors can be

seen on Table 5.2, 5.3 and 5.4 below. A total of 23 security design patterns were defined by Steel, et.al (2005). These were classified into four logical tiers consisting of the web tier, the business tier, web services and identity tier. Table 5.2 show the security design patterns defined under each logical tier:

Table 5. 2: Security Design Pattern by Steel, et.al (2005)

S\no	Web Tier	Business Tier	Web Services	Identity Tier
1	Authentication Enforcer	Audit Interceptor	Message Inspector Gateway	Assertion Builder
2	Authorization Enforcer	Container Managed Security	Secure Message Router	Credential Tokenizer
3	Intercepting Validator	Dynamic Service Management	Message Inspector	Single Sign On (SSO) Delegator
4	Secure Base Action	Obfuscated Transfer Object		Password Synchronizer
5	Secure Pipe	Policy Delegate		
6	Secure Service Proxy	Secure Service Façade		
7	Secure Session Manager	Secure Session Object		
8	Intercepting Web Agent			
9	Secure logger			

A total of 13 security design patterns were defined by Blakley, et.al (2004) and these were classified into two groups. This consisted of the available security design patterns and the protected security design patterns. Table 5.3 shows the security design patterns classified under each category.

Table 5. 3: Security Design Patterns by Blakley, et.al (2004)

s\no	Available	Protected
1	Check pointed System	Protected System
2	Standby	Policy
3	Comparator- Check Fault – Tolerant System	Authenticator
4	Replicated System	Subject Descriptor
5	Error/ Detection/Correction	Secure Communication
6		Security Context
7		Security Association
8		Secure Proxy

The security design patterns defined by Kienzle and Elder (2003) were also classified into two categories. These include the structural patterns and procedural patterns. The structural patterns consist of 13 main security design patterns and 3 mini-patterns. The mini-patterns are less formal and shorter discussion that were included as a supplement to the main security design patterns. The procedural patterns consist of 13 security design patterns. Table 5.4 shows the security design patterns that were classified under each category.

Table 5. 4: Security Design Patterns by Kienzle and Elder (2003)

s\no	Structural Patterns	Procedural Pattern
1	Account Lockout	Build Server From Ground up
2	Authenticated Session	Choose the right stuff
3	Client Data Storage	Document Security goals
4	Client Input filters	Document Server Configuration
5	Directed Session (M)	Enrol by Validating out of band
6	Hidden Implementation (M)	Enrol Using Third party Validation
7	Encrypted Storage	Enrol with pre-existing Shared Secret
8	Minefield	Enrol without validating
9	Network Address Blacklist	Log for Audit
10	Partitioned Application	Patch Proactively
11	Password Authentication	Red Team the Design
12	Password Propagation	Share responsibility for security
13	Secure Assertion	Test on a staging server
14	Server Sandbox	
15	Trusted Proxy	
16	Validated Transaction	

There are other security design patterns that have been defined by other authors different from the ones highlighted above (See chapter 2 for further discussion). For this reason, a decision had to be made on which security design patterns to be analysed for abstracting the data needed for training the neural network. The decision to use the security design patterns defined by Steel, et.al (2005), Blakley, et.al (2004) and Kienzle and Elder (2003) base on the following reasons:

1. Security design pattern by Blakley, et.al (2004) was initiated by the Open Group Security Forum in a coordinated effort to resolve the problem of lack of clear definition of security design patterns. The security design patterns were defined based on a comprehensive list of existing security design pattern to be used as a guide by software developers.
2. There is an existing research work by Halkidis, S.T. et al. (2006) in which the security design by Blakley, et.al (2004) was analysed qualitatively which provided insight into this research work (See section 3 further discussion)

3. Since security design patterns have been defined for different purposes, the security design patterns by Steel, et.al (2005) and Kienzle and Elder (2003) were chosen because they both address web related security issues.

5.3. Data Encoding

In order to encode the data needed for training the second neural network, the collected data were initially analysed. Since the data to be used as input for the second neural network is based on the regularly expressed attack pattern identified by the first neural network, no further analysis was required because the data was already analysed (See chapter 4 for more discussion) and classified according to the type of threat they represent as shown in Table 5.1. The information from the analysis on the attack components in the regularly expressed attack patterns was used to encode the input data. Table 5.5 shows the attributes of the regularly expressed attack patterns used in encoding the input data to the second neural network (See Chapter 3 section 4.2 for further discussion).

Table 5. 5: Attributes of Regularly Expressed Attack Patterns

s\no	Attribute	Observable	Value
1	Attack ID	Attack Pattern	Attack ID
2	Resource Attacked	Attack Component	Attack Component ID
3	Attack Vector	Attack Component	Attack Component ID
4	Attack Type	Availability	1
		Integrity	2
		Confidentiality	3

The taxonomy of security design patterns by Wiesauer and Sametinger (2009) was based on the description of the attack patterns in Common Attack Pattern Enumeration and Classification (CAPEC) catalogue and the intent and purpose of the security design patterns. The authors stated that since the classification of the attack patterns in CAPEC catalogue is based on STRIDE, their proposed taxonomy on security design patterns could be considered as classification based on STRIDE as well. Building on this approach, the security design patterns defined by Steel, et.al (2005), Blakley, et.al (2004) and Kienzle and Elder (2003) in section 2 were analysed. From previous research by Halkidis, S.T. et al. (2006), it was observed that security design pattern by Blakley, et.al (2004) was analysed qualitatively using Microsoft threat classification (STRIDE) to find out the security design pattern that provides protection on each of the threat category. The result of the analysis is shown on Table 5.6 and was used as part of the data needed for training the second neural network.

Table 5. 6: Classification of Security Design Pattern by Blakley, et.al (2004)

s\no	Security Pattern	S	T	R	I	D	E
1	Check pointed System					X	

2	Standby					X	
3	Comparator- Check Fault – Tolerant System					X	
4	Replicated System					X	
5	Error/ Detection/Correction					X	
6	Protected System	X	X		X		X
7	Policy	X	X		X		X
8	Authenticator	X	X		X		X
9	Subject Descriptor						
10	Secure Communication	X	X		X	X	X
11	Security Context		X		X		X
12	Security Association	X	X		X		X
13	Secure Proxy	X	X		X		X

In a similar manner, security design patterns defined by Steel, et.al (2005), and Kienzle and Elder (2003) were analysed. During the analysis of security design patterns by Kienzle and Elder (2003), the procedural patterns were not analysed because they consisted security patterns which were not implemented in the software application. Procedural patterns were defined for the purpose of improving the development process of mission critical software applications. They can impact the management of a software development project when adopted by software developers. Table 5.7 and Table 5.8 show the threat category that the security design patterns were classified after the analysis. It would be noticed that the secure assertion and dynamic service management on Table 5.7 and Table 5.8 respectively were not classified under any category. These security design patterns are related to each other and could not be classified under any threat category because they only provide monitoring and reporting of the system events but do not offer protection against STRIDE attacks

Table 5. 7: Classification of Security Design Pattern by Kienzle and Elder (2003)

s\no	Structural Patterns	S	T	R	I	D	E
1	Account Lockout	X	X		X		X
2	Authenticated Session	X	X		X		X
3	Client Data Storage				X		X
4	Client Input filters		X		X	X	X
5	Directed Session (M)		X				
6	Hidden Implementation (M)				X		
7	Encrypted Storage				X		X
8	Minefield	X		X			
9	Network Address Blacklist	X				X	
10	Partitioned Application						X
11	Password Authentication	X	X		X		X
12	Password Propagation	X	X		X		X
13	Secure Assertion						

14	Server Sandbox		X		X		X
15	Trusted Proxy	X	X		X		X
16	Validated Transaction		X				

Table 5. 8: Classification of Security Design Pattern by Steel, et.al (2005)

s\no	Security Pattern	S	T	R	I	D	E
1	Authentication Enforcer	X	X		X		X
2	Authorization Enforcer	X	X		X		X
3	Intercepting Validator		X		X		X
4	Secure Base Action	X	X		X	X	X
5	Secure Pipe	X	X		X	X	X
6	Secure Service Proxy	X	X		X		X
7	Secure Session Manager	X	X		X		X
8	Intercepting Web Agent	X	X		X		X
9	Secure logger		X	X	X		
10	Audit Interceptor		X	X	X		
11	Container Managed Security	X	X		X		X
12	Dynamic Service Management						
13	Obfuscated Transfer Object				X		
14	Policy Delegate				X	X	
15	Secure Service Façade	X	X		X		X
16	Secure Session Object	X			X		X
17	Message Inspector Gateway	X	X	X	X		X
18	Secure Message Router	X			X		
19	Message Inspector	X	X	X	X		
20	Assertion Builder	X					
21	Credential Tokenizer	X		X			
22	Single Sign On (SSO) Delegator	X			X		
23	Password Synchronizer	X			X		

Following the analysis of the data collected on the regularly expressed attack patterns and the security design patterns, the data needed for training the second neural network was encoded. A total of 226 training data samples were abstracted from the regularly expressed attack patterns using the attributes in Table 5.5. To encode the data, the corresponding value for the information abstracted by each attribute in the Table was used in encoding the data. For instance, regularly expressed attack pattern 1 is represented as:

(User⁺) (Server⁺) (Log⁺) (HardDrive⁺)

Based on the analysis of the data collected on this attack pattern, the information on Table 5.9 was abstracted from attack pattern 1 using the regularly expressed attack pattern attributes.

Table 5. 9: Sample of Pre-processed training data from attack pattern

Attack ID	Resource Attacked	Attack Vector	Attack Type
1	Hard Drive	Log	Availability

In order to encode the information abstracted in the Table, the attack component ID for Hard Drive and Log was used for their encoding (See Table 4.2 in chapter 4 Section 3) The corresponding values for Availability and Denial of Service in Table 5.5 was also used for their encoding. Table 5.10 below shows the training data for the example above after it has been encoded.

Table 5. 10: Sample of training data after encoding

Attack ID	Resource Attacked	Attack Vector	Attack Type
1	42	58	1

The next stage involves converting the encoded data into ASCII comma delimited format which can be used to train the neural network as shown below

1, 42, 58, 1

The data is then loaded into the neural network for training as shown in the following Table.

Table 5. 11: Sample of input data into neural network

Input 1	Input 2	Input 3	Input 4
1	42	58	1

For the expected output, security design patterns by Steel, et.al (2005), and Kienzle and Elder (2003) were grouped into six groups with respect to STRIDE. Each group provides possible solutions to the threats identified under each threat category of STRIDE. A unique ID is assigned each group so that the neural network can match them to the corresponding attack patterns. Based on this encoding, the neural network is expected to identify the possible solution for the attack pattern in the Table 5.11 above by giving the following output:

1, 0, 0, 0, 0, 0

Table 5.12 -5.17 shows the six groups the security design patterns were classified into using STRIDE

Table 5. 12: Security Design Patterns Group 1

s\no	Security patterns by Steel, et.al, (2005)	Security Patterns by Blakley, et.al (2004)	Security patterns by Kienzle and Elder (2003)
1	Authentication Enforcer	Protected System	Account Lockout
2	Authorization Enforcer	Policy	Authenticated Session
3	Secure Base Action	Authenticator	Minefield
4	Secure Pipe	Secure Communication	Network Address Blacklist
5	Secure Service Proxy	Security Association	Password Authentication
6	Secure Session Manager	Secure Proxy	Password Propagation
7	Intercepting Web Agent		Trusted Proxy
8	Container Managed Security		
9	Secure Service Façade		
10	Secure Session Object		
11	Message Inspector Gateway		
12	Secure Message Router		
13	Message Inspector		
14	Assertion Builder		
15	Credential Tokenizer		
16	Single Sign On (SSO) Delegator		
16	Password Synchronizer		

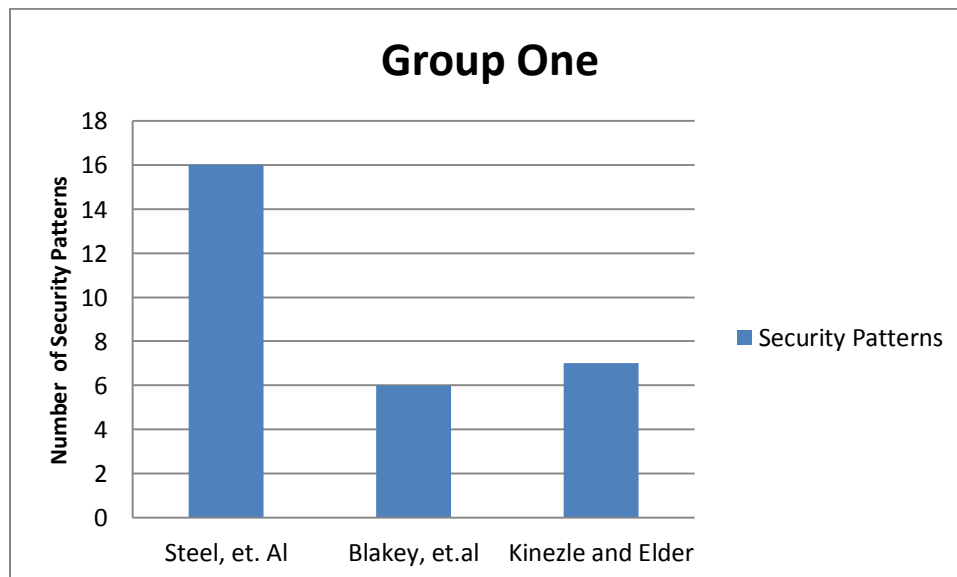


Figure 5. 1: Security Design Patterns Group 1

Table 5. 13:Security Design Patterns Group 2

s\no	Security patterns by Steel, et.al, (2005)	Security Patterns by Blakley, et.al (2004)	Security patterns by Kienzle and Elder (2003)
1	Authentication Enforcer	Protected System	Account Lockout
2	Authorization Enforcer	Policy	Authenticated Session
3	Intercepting Validator	Authenticator	Client Input filters
4	Secure Base Action	Secure Communication	Directed Session (M)
5	Secure Pipe	Security Context	Password Authentication
6	Secure Service Proxy	Security Association	Password Propagation
7	Secure Session Manager	Secure Proxy	Server Sandbox
8	Intercepting Web Agent		Trusted Proxy
9	Secure logger		Validated Transaction
10	Audit Interceptor		
11	Container Managed Security		
12	Secure Service Façade		
13	Message Inspector Gateway		
14	Message Inspector		

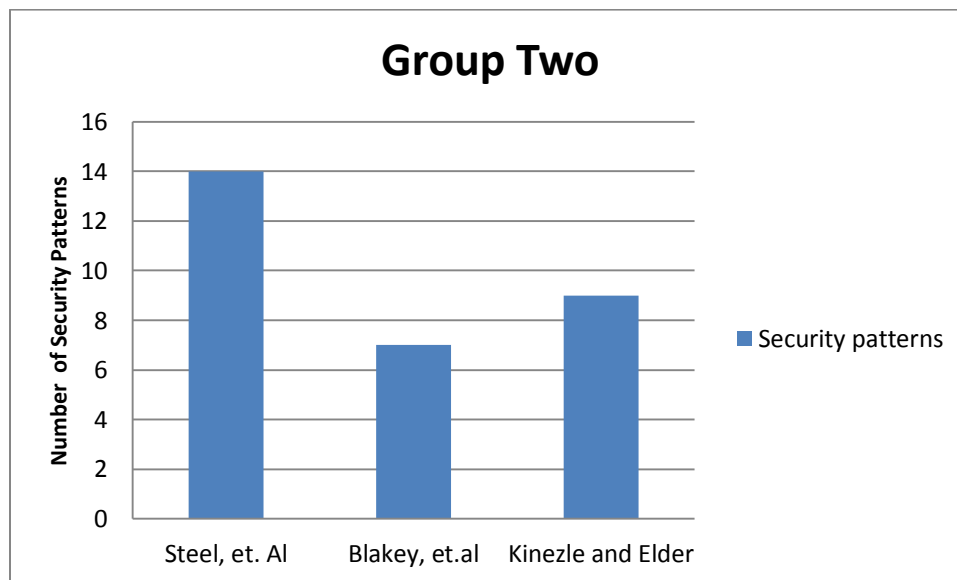


Figure 5. 2: Security Design Patterns Group 2

Table 5. 14: Security Design Patterns Group 3

s\no	Security patterns by Steel, et.al, (2005)	Security Patterns by Blakley, et.al (2004)	Security patterns by Kienzle and Elder (2003)
1	Secure logger		Minefield
2	Audit Interceptor		
3	Message Inspector		

	Gateway		
4	Message Inspector		
5	Credential Tokenizer		

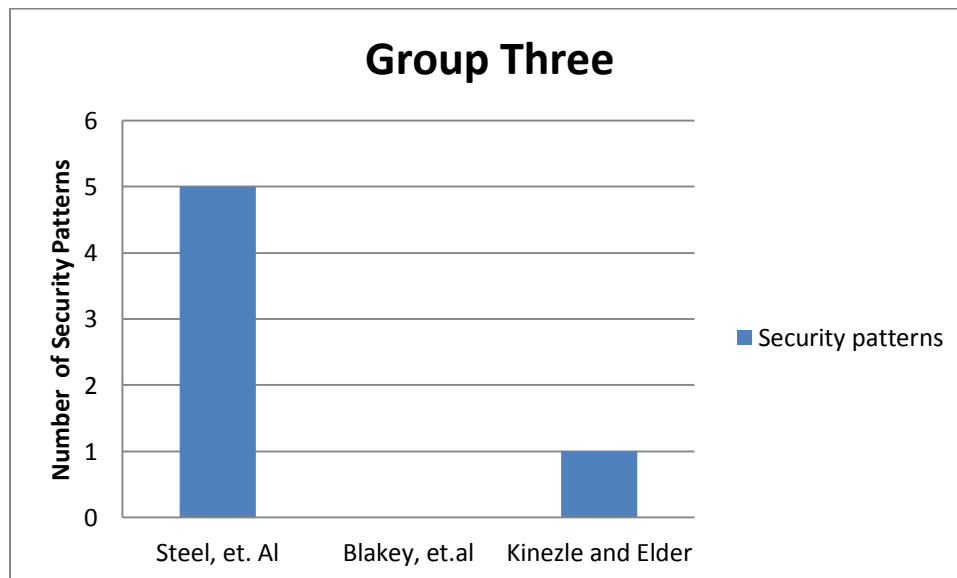


Figure 5. 3: Security Design Patterns Group 3

Table 5. 15: Security Design Patterns Group 4

s\no	Security patterns by Steel, et.al, (2005)	Security Patterns by Blakley, et.al (2004)	Security patterns by Kienzle and Elder (2003)
1	Authentication Enforcer	Protected System	Account Lockout
2	Authorization Enforcer	Policy	Authenticated Session
3	Intercepting Validator	Authenticator	Client Data Storage
4	Secure Base Action	Secure Communication	Client Input filters
5	Secure Pipe	Security Context	Hidden Implementation (M)
6	Secure Service Proxy	Security Association	Password Propagation
7	Secure Session Manager	Secure Proxy	Password Authentication
8	Intercepting Web Agent		Password Propagation
9	Secure logger		Server Sandbox
10	Audit Interceptor		Trusted Proxy
11	Container Managed Security		
12	Obfuscated Transfer Object		
13	Policy Delegate		
14	Secure Service Façade		
15	Secure Session Object		
16	Message Inspector Gateway		
17	Secure Message Router		

18	Message Inspector		
19	Single Sign On (SSO) Delegator		
20	Password Synchronizer		

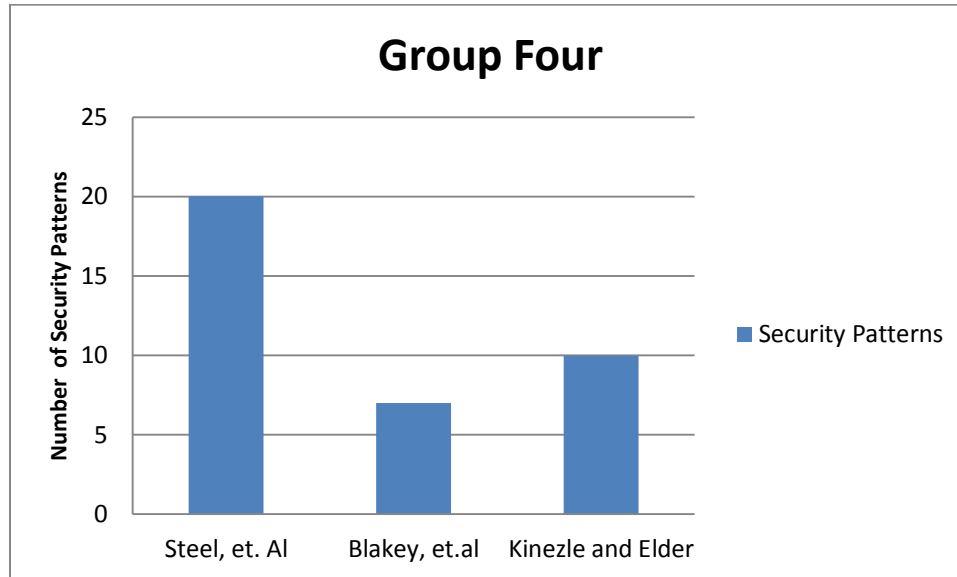


Figure 5. 4: Security Design Patterns Group 4

Table 5. 16: Security Design Patterns Group 5

s\no	Security patterns by Steel, et.al, (2005)	Security Patterns by Blakley, et.al (2004)	Security patterns by Kienzle and Elder (2003)
1	Secure Base Action	Check pointed System	Client Input filters
2	Secure Pipe	Standby	Network Address Blacklist
3	Policy Delegate	Comparator- Check Fault – Tolerant System	
4		Replicated System	
5		Error/ Detection/Correction	
6		Secure Communication	

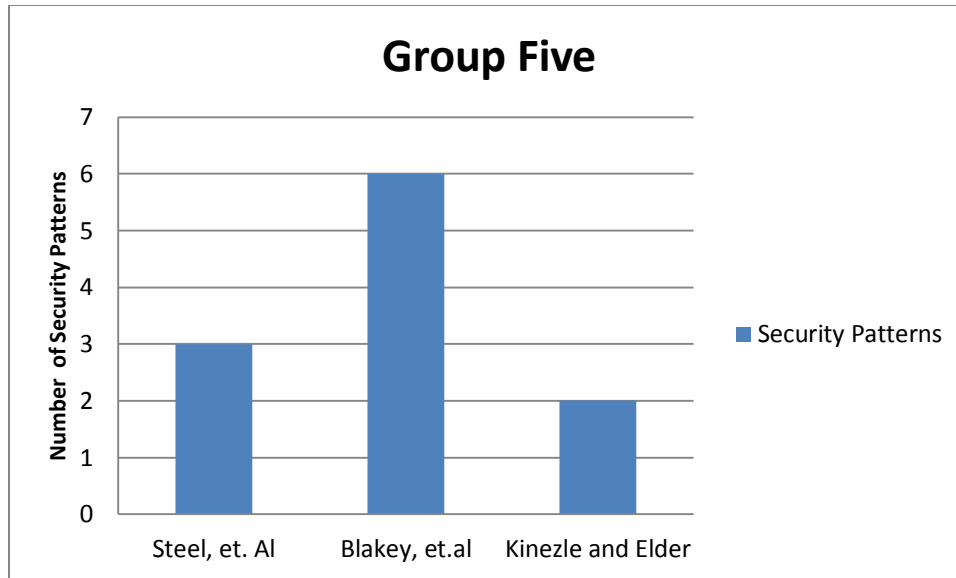


Figure 5. 5: Security Design Patterns Group 5

Table 5. 17: Security Design Patterns Group 6

s\no	Security patterns by Steel, et.al, (2005)	Security Patterns by Blakley, et.al (2004)	Security patterns by Kienzle and Elder (2003)
1	Authentication Enforcer	Protected System	Account Lockout
2	Authorization Enforcer	Policy	Authenticated Session
3	Intercepting Validator	Authenticator	Client Data Storage
4	Secure Base Action	Secure Communication	Client Input filters
5	Secure Pipe	Security Context	Encrypted Storage
6	Secure Service Proxy	Security Association	Partitioned Application
7	Secure Session Manager	Secure Proxy	Password Authentication
8	Intercepting Web Agent		Password Propagation
9	Container Managed Security		Server Sandbox
10	Secure Service Façade		Trusted Proxy
11	Secure Session Object		
12	Message Inspector Gateway		

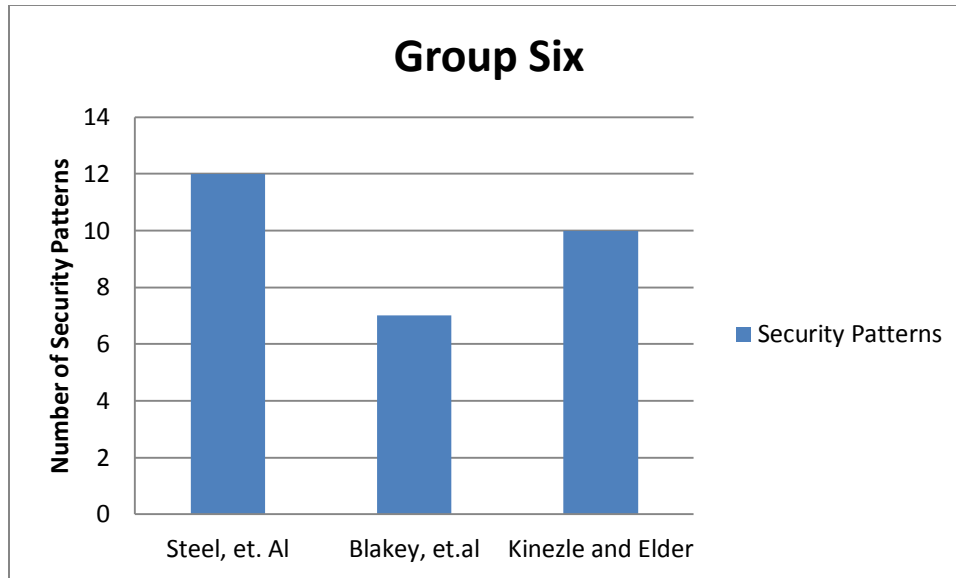


Figure 5. 6: Security Design Patterns Group 6

5.4. Neural Network Architecture

A feed-forward back-propagation neural network is used to analyse the attack patterns and generate possible solutions from the security design patterns that can be help in mitigating the threat identified in the attack patterns. The architecture of the second neural network is the same as that of the first neural network. This is the standard three layer neural network architecture consisting of the input layer, the hidden layer and the outer layer. To optimize the performance of the neural network, LM, RP and SCG training optimization algorithms were also applied in the same way as the first neural network (See chapter four for further discussion) With respect to the transfer functions, a tan-sigmoid transfer function was applied to the various connection weights in the hidden nodes and in the output nodes, the linear transfer function is applied to its weight. Since a supervised learning architecture (i.e. back-propagation) was adopted for the second neural network, the data discussed in section 3 was used for its training.

5.5. Neural Network Training

To train the second neural network the training data set is divided into two sets. The first set of data is the training data sets (201 Samples) that were presented to the neural network during training. The second set (26 Samples) is the data that were used to test the performance of the neural network after it had been trained. In a similar way as the first neural network, the performance of the second neural network was observed when 80, 90, 100, 110 and 120 hidden neurons were used for each training optimization algorithm that was applied. The training performance is measured by Mean Squared Error (MSE) and the training stops when the generalization stops improving or when the 1500th iteration is reached. Mat lab Neural

Network tool box was used to perform the training. The training parameters also include the learning rate which is set to 0.01 with a goal of 0; maximum fail set to 6 and a minimum gradient of 0.000001

5.6. Performance Analysis and Discussion

In a similar way to neural network I, different numbers of neurons were applied to the neural network to further optimize its performance. The training was executed in five simulations to obtain the average results of its performance because the neural network is initiated with random weights during its training and this gives different results. Therefore, following the same process used in the analysis of the performance results from neural network I, the average results on the training time, MSE and number of epoch were also used in analysis of the performance of the neural network II. The result of the performance of the neural network II when LM training optimization algorithm is applied is also not presented in this section because the performance neural network was poor when the training algorithm was applied. It took an average of 2 minutes to complete its training which was faster than the time observed when it was used in training neural network I. The lowest MSE obtained during its training was 0.659 which is very far from the set goal in training the network. A statistical analysis was also carried out to establish the significance of the training optimization algorithms applied to neural network II using statistical tools.

5.6.1. Mean Square Error (MSE)

Table 5.18 and 5.19 show the performance of the neural network II when SCG and RP training optimization algorithms were applied to the network based on their MSE results. The lowest MSE result (0.001838) obtained when SCG training optimization algorithms was applied to the network with 100 neurons. The highest MSE result was observed when 90 neurons were implemented in the network. It was also observed that the MSE results were below 0.0044 for the network could closely match the attack patterns to the expected security design patterns. The performance of the network was tested with the 25 test data sample. The output generated from the network was identical to the expected output. Therefore the performance of neural network II when SCG training optimization algorithm was applied was considered to be very good. Figure 5.7 shows the plot of the MSE result for neural network II during training. Figure 5.8 shows the plot of the average MSE result for the different number of neurons applied to the network.

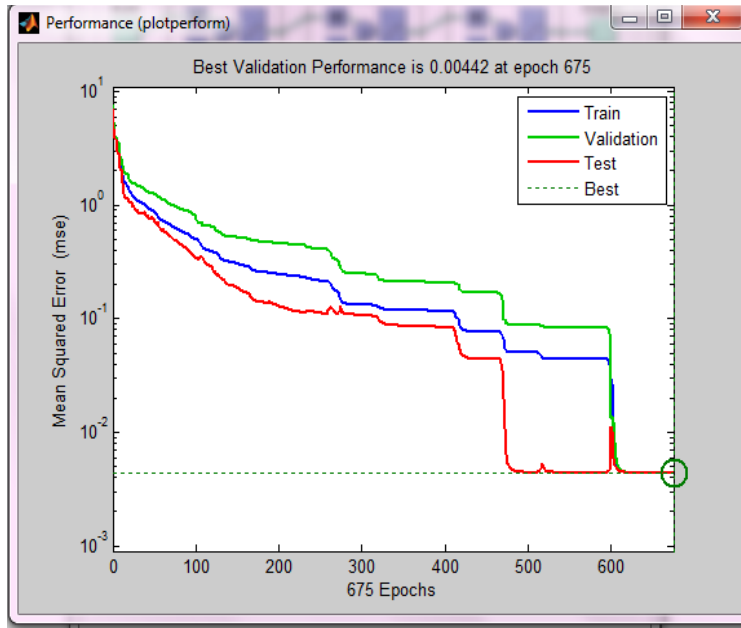


Figure 5. 7: Plot of MSE for Neural Network II

Table 5. 18: MSE of Neural Network II with SCG Applied

s\no	Number of Hidden Neurons				
	80	90	100	110	120
1	0.0029	0.00332	0.00124	0.00347	0.00319
2	0.000829	0.00126	0.00166	0.00159	0.00264
3	0.00455	0.00376	0.00265	0.00124	0.00191
4	0.00333	0.0029	0.000829	0.00385	0.00124
5	0.00249	0.00438	0.00281	0.00373	0.00145
Ave	0.00282	0.003124	0.001838	0.002776	0.002086

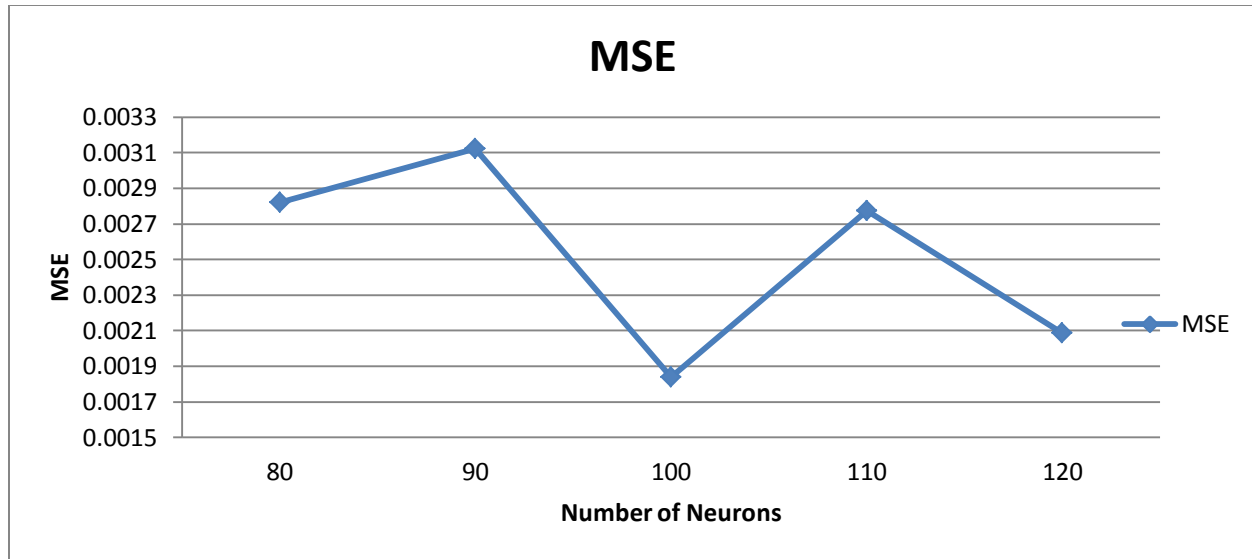


Figure 5. 8: Number of Epoch used in Neural Network I with SCG Applied

The performance of neural network II was better when RP training optimization algorithm was applied to the network. It was observed that the MSE results obtained were lower than the MSE results obtained with SCG training algorithm was applied to the network. The highest MSE (0.002286) was obtained when 110 neurons were implemented in the network and the lowest (0.001055) was obtained when 90 neurons were implemented. The performance of the network was also tested with the test data sample and this generated an output which was identical to the expected output (see chapter 6 for more discussion). Figure 5.9 shows the plot of the average MSE results obtained

Table 5. 19: MSE of Neural Network I with RP Applied

s\no	Number of Hidden Neurons				
	80	90	100	110	120
1	0.000323	0.0019	0.000522	0.00115	0.00394
2	0.00385	0.00138	0.0038	0.00488	0.000757
3	0.000376	0.000776	0.000324	0.000969	0.000537
4	0.001	0.00114	0.000316	0.00147	0.000107
5	0.000367	0.0000789	0.00351	0.00296	0.000509
Ave	0.001183	0.001055	0.001694	0.002286	0.00117

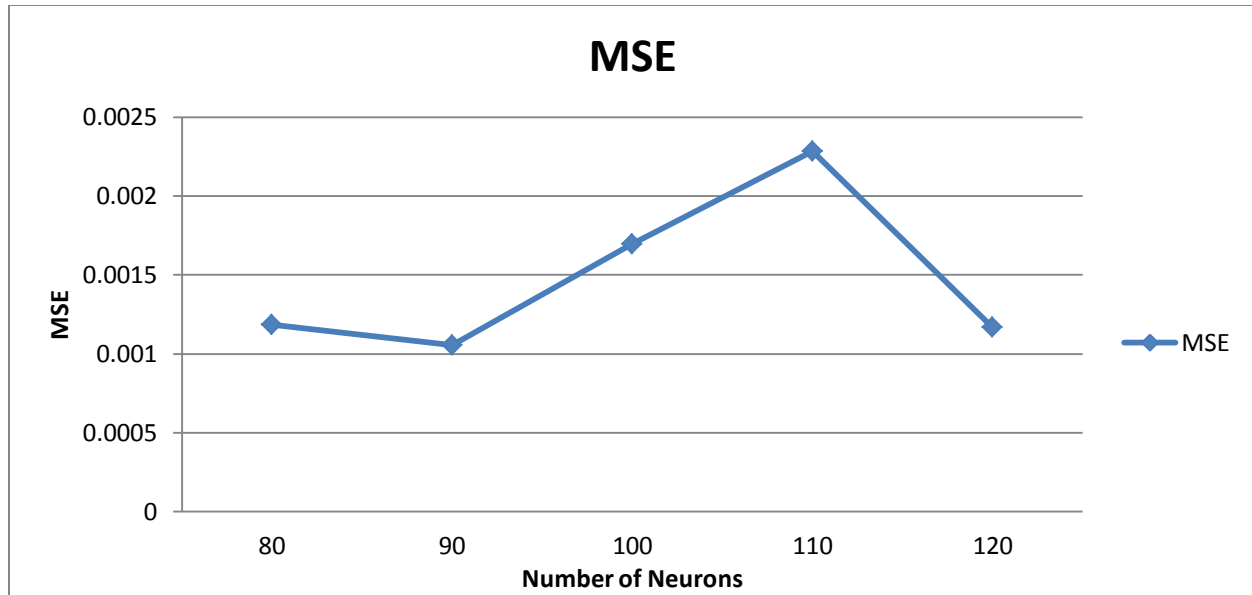


Figure 5. 9:MSE of Neural Network II with RP Applied

5.6.2. Number of Epochs

Table 5.20 and 5.21 shows show the performance of the first neural network based on number of epochs used in training the network. The highest number of average epoch used when SCG was applied as the training optimization algorithm was 532.4 with 100 neurons implemented. This dropped to the lowest (364.2) when 110 neurons were implemented in the network. Figure 5.10 shows the plot of the average number of epoch used in training with different number of neurons.

Table 5. 20: Number of Epoch used in Neural Network II with SCG Applied

s\no	Number of Hidden Neurons				
	80	90	100	110	120
1	582	512	636	281	345
2	535	408	686	302	312
3	257	410	352	370	385
4	315	518	626	302	659
5	722	408	362	566	574
Ave	482.2	451.2	532.4	364.2	455

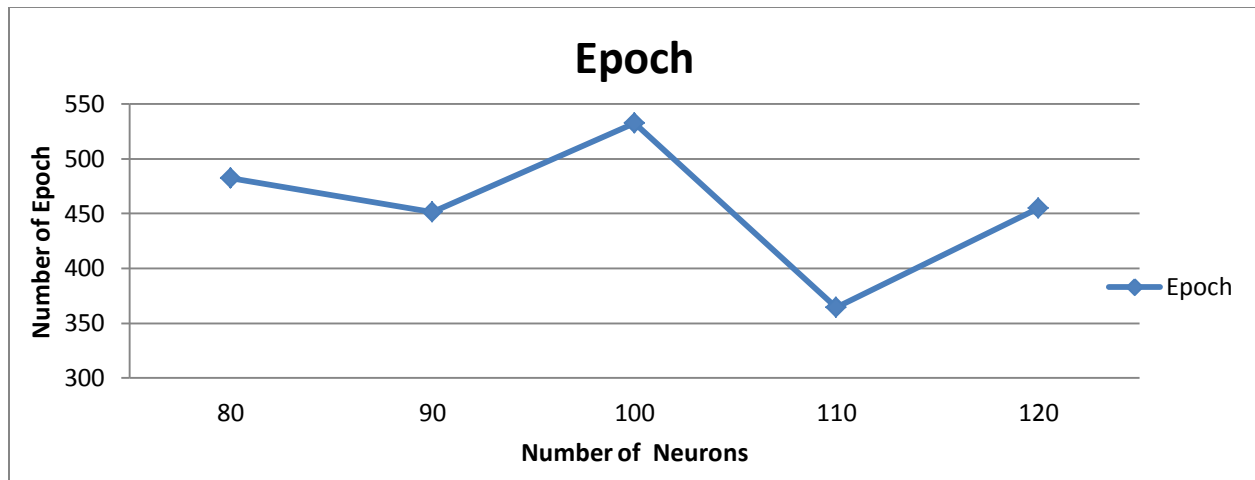


Figure 5. 10: Number of Epoch used in Neural Network I with SCG Applied

It was observed that when RP training optimization algorithm was applied to the network, the number of epoch used was greater than when SCG training optimization algorithm was applied. The highest number of epoch used with RP training optimization algorithm is 1324.8 when 80 neurons were implemented and the lowest was 906.8 when 110 neurons were implemented. The plot of the average number of epoch used when RP training optimization algorithm was applied in Figure 5.11 shows that the number of epochs used decreases as the number of neurons implemented increases from 80 to 110 and increased slightly when the number of neurons was increased to 120.

Table 5. 21: Number of Epoch used in Neural Network II with RP Applied

s\ no	Number of Hidden Neurons				
	80	90	100	110	120
1	1500	1448	1005	1101	610
2	1038	845	998	635	987
3	1500	1245	989	827	980
4	1086	1188	1133	983	1009
5	1500	1500	1056	988	993
Ave	1324.8	1245.2	1036.2	906.8	915.8

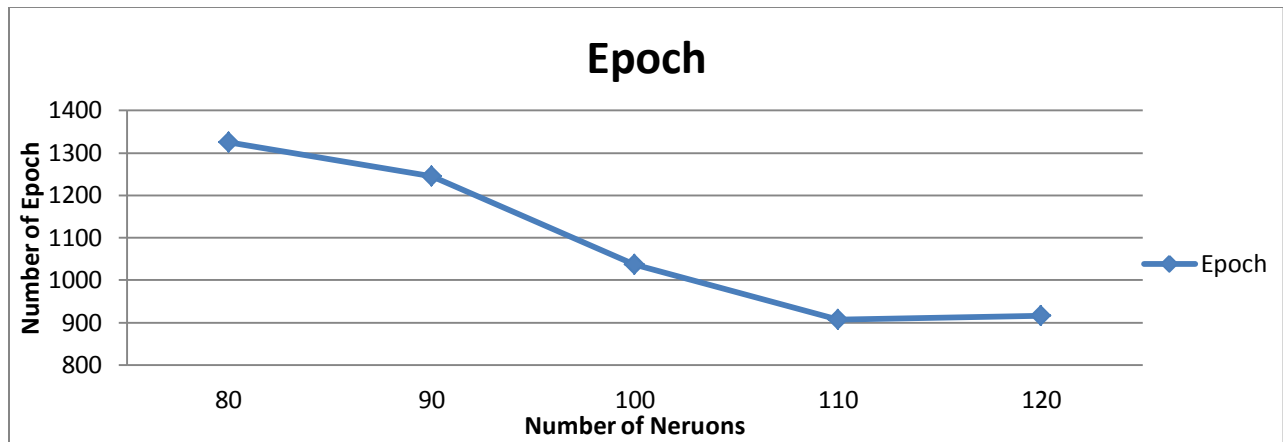


Figure 5. 11: Number of Epoch used in Neural Network II with RP Applied

5.6.3. Training Time

Table 5.22 and 5.23 show the time spent in training neural networks II with SCG and RP training optimization algorithms respectively. The plot of the average time spent in training the network when SCG RP training optimization algorithm was applied (Figure 5.12) shows a sharp increase in the training time from 20 seconds to 45 seconds when the number of neurons implemented increased from 110 to 120.

Table 5. 22: Training Time for Neural Network II with SCG Applied

s\no	Number of Hidden Neurons				
	80	90	100	110	120
1	00:25	00:23	00:30	00:17	00:34
2	00:22	00:29	00:34	00:16	00:30
3	00:11	00:18	00:17	00:20	00:38
4	00:13	00:23	00:32	00:17	01:05
5	00:30	00:19	00:17	00:31	00:57
Ave	00:20	00:22	00:26	00:20	00:45

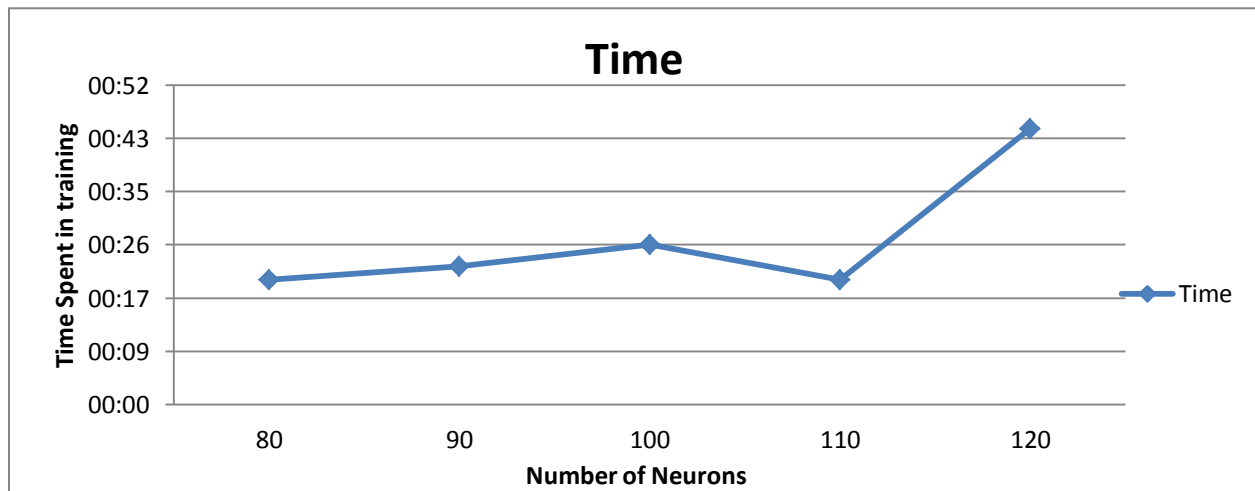


Figure 5. 12: Training Time for Neural Network II with SCG Applied

Compared to the time spent when SCG training optimization algorithm was applied to the network, the time spent in training the network when RP training optimization algorithm was applied to the network was longer. The plot of the average time spent in training the network as shown in figure 5.13 reveal that the shortest time spent in training (47 seconds) was when 90 neurons was implemented in the network and the longest time spent in training the network (1 minutes and 1 second) was when 80 neurons was implemented. It would be noticed that the time spent in training the network when RP training optimization algorithm was applied decreased as the number of neurons implemented increased from 80 to 110 and increased slightly when 120 neurons were implemented.

Table 5. 23: Training Time for Neural Network II with RP Applied

s\no	Number of Hidden Neurons				
	80	90	100	110	120
1	01:32	01:05	00:48	00:54	00:31
2	00:44	00:38	00:46	00:34	00:52
3	01:01	00:54	00:46	00:43	00:50
4	00:44	00:51	00:52	00:52	00:59
5	01:02	01:05	00:49	00:51	01:01
Ave	01:01	00:55	00:48	00:47	00:51

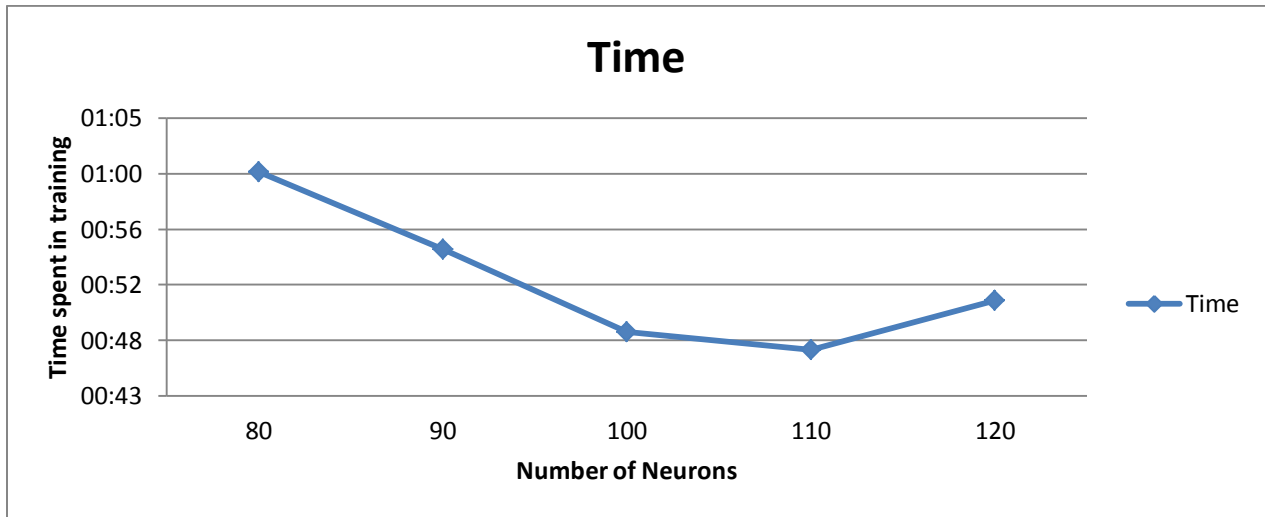


Figure 5. 13: Training Time for Neural Network II with RP Applied

6. Summary of Chapter 5

The Implementation of the second neural network has been demonstrated in this chapter. Using the data from the analysis of the regularly expressed attack pattern and security design pattern data needed for the training of the neural network was abstracted. The list of attributes used in abstracting the data was presented and the encoding of the data was demonstrated. Similar neural network architecture to the first neural network was adopted for the second neural network architecture. LM, RP and SCG training optimization algorithms was also applied

to neural network and its performance was measured by Mean Square Error (MSE). The overall result of the network shows that neural network II performs better when it was trained with RP training optimization algorithm. The best MSE performance was observed when 90 neurons were implemented in the network and the training time lasted for 55 seconds. Based on this result, 90 hidden neurons were chosen for the implementation of neural network II and RP training optimization algorithm was chosen for its training. Similarly, neural network II was tested using test data sample and output generated was identical to the expected security design pattern. In the next chapter, the validation study is presented to analyse how close the output of the proposed neural network tool produces matches the expected output. A case study is also presented to demonstrate how the neural network tool can be used to integrate security into software design.

Chapter 6. Result and Discussion

6.1. Introduction

Having analyzed the performance of neural network I and II in the previous chapter, in this chapter a statistical analysis is conducted to compare the performance of the networks when SCG and RP training optimization algorithms were applied. This is followed by a validation study in which the test data from chapter 4, section 5 and chapter 5, section 5 is used to test the performance of neural network I and II respectively. The results of the actual output of the networks are compared to their expected output. Their difference are identified and analyzed. The performance of neural network I was also compared to the performance of students conducting SAFE-T in Gegick and Williams' feasibility study. To demonstrate how the proposed neural tool can be used in integrating security into software design, a case study on a real life system is presented.

6.2. Statistical Analysis on the Performance of Neural Network I

To compare the performance of the neural network I when SCG and RP training optimization algorithms were applied to the network a statistical analysis was conducted. The one way analysis of variance (ANOVA) was used to compare the average MSE results obtained when the two training algorithms were applied to the network. Table 6.1 shows the hypotheses that were proposed in the analysis.

Table 6. 1:Hypothesis proposed for comparing performance of neural network I

Hypothesis	Test	Explanation
H_o	$\mu_1 = \mu_2$	The mean of MSE results obtained from the network when the two training algorithms are applied are the same
H_A	$\mu_1 \neq \mu_2$	The mean of MSE results are different

To carry out the analysis, the average results of the MSE obtained was processed by multiplying each value by 1000 to make it easier for the calculation to be computed. Table 6.2 shows the initial and final data after it was processed. The decision rule for accepting or rejecting the proposed hypothesis given the two training algorithms each with five different numbers of neurons is based on the critical F- distribution value (F_{crt}) of 5.32 with 1 degree of freedom (DF)(i.e. between the groups) and 8 DF (i.e. within the groups) and at 0.05 confidence level interval. If the observed F statistic value (F_{obs}) is lesser than F_{crt} value (i.e. $F_{obs} < F_{crt}$) the null hypothesis H_o is accepted otherwise it is rejected in favour of H_A . Table 6.3 is the ANOVA Table used in calculating the F_{obs} .

Table 6. 2:Average of MSE Results of neural network implemented with SCG and RP

Number of Neurons	SCG		RP	
	Initial	Final	Initial	Final
80	0.003246	3.246	0.003422	3.422
90	0.002148	2.148	0.00284	2.84
100	0.002469	2.469	0.003496	3.496
110	0.00239	2.39	0.003528	3.528
120	0.002812	2.812	0.003168	3.168

Table 6. 3:ANOVA Table for Average MSE Result for Neural Network I

Source	DF	SS	MS	F	P
Treatment	1	1.1486	1.1486	8.6621	0.05
Error	8	1.0608	0.1326		
Total	9	2.2094			

Result from the ANOVA Table show that the $F_{obs} = 8.6621$ and since this is greater than F_{crt} (5.32), we reject the null hypothesis and conclude that the average MSE results obtained from neural network I when the two training algorithms are applied to the network are significantly different. Furthermore based on the analysis of the MSE in chapter 4, section 6.1, neural network I had a better performance when SCG training algorithm was applied to the network.

6.3 Statistical Analysis on the Performance of Neural Network II

Also using statistical analysis, the performance of neural network II when SCG and RP training optimization algorithms were applied to the network was analyzed. The one way analysis of variance was used to compare the average MSE results obtained from the network. Table 6.4 shows the proposed hypotheses for the analysis.

Table 6. 4:Hypothesis proposed for comparing performance of neural network II

Hypothesis	Test	Explanation
H_o	$\mu_1 = \mu_2$	The mean of MSE results obtained from neural network II when the two training algorithms are applied are the same
H_A	$\mu_1 \neq \mu_2$	The mean of MSE results are different

The average results of the MSE obtained from neural network II was also processed in a similar way to section 2.4 by multiplying each value by 1000 to make it easier for the statistical calculation to be computed. Table 6.5 shows the initial and final data after it was processed. F_{crt} value of 5.32 with 1 and 8 DF at 0.05 confidence level interval was used as the decision rule for either accepting or rejecting the null hypothesis. Therefore if $F_{obs} < F_{crt}$, then the null hypothesis H_o is accepted. Otherwise it is rejected in favour of H_A . Table 6.6 is the ANOVA Table used in calculating the F_{obs} .

Table 6. 5:Average of MSE Results of Neural Network II implemented with SCG and RP

Number of Neurons	SCG		RP	
	Initial	Final	Initial	final
80	0.00282	2.82	0.001183	1.183
90	0.003124	3.124	0.001055	1.055
100	0.001838	1.838	0.001694	1.694
110	0.002776	2.776	0.002286	2.286
120	0.002086	2.086	0.00117	1.17

Table 6. 6:ANOVA Table for Average MSE Result for Neural Network II

Source	DF	SS	MS	F	P
Treatment	1	2.7626	2.7626	8.1385	0.05
Error	8	2.2337	0.2792		
Total	9	4.9963			

Since $F_{obs} > F_{crt}$, the null hypothesis is rejected in favour of H_A and conclude that the average MSE results obtained from neural network II when the two training algorithms are applied to the network are significantly different. By comparing the average MSE results obtained from the network, it can be established that neural network II had a better performance when RP training algorithm was applied to the network.

6.4. Validation Study

In the validation study, the test data from the data collected was used to test the performance of the neural networks after they had been trained. The expected and actual output of the network and then compared and analyzed.

6.4.1 Neural Network I

For neural network I, a total of 52 data samples were used to test the network. Since neural network I involve two networks, the test data was divided into two data sets each consisting of 26 data samples corresponding to each neural network (See Table 4.6, chapter 4 section 5). Table 6.1 shows the result of the actual output of the neural network I and its expected output.

Table 6. 7: Actual and expected output of Neural Network I

s\n	Attack Pattern Investigated	Actual Output	Expected Output
Results from Network 1			
1	Attack Pattern 1	0.9970	1
2	Attack Pattern 2	1.5821	2
3	Attack Pattern 3	3.0000	3
4	Attack Pattern 4	3.9991	4
5	Attack Pattern 5	4.9913	5
6	Attack Pattern 6	6.0000	6
7	Attack Pattern 7	6.9998	7
8	Attack Pattern 8	7.9995	8
9	Attack Pattern 9	8.9972	9
10	Attack Pattern 10	9.9189	10

11	Attack Pattern 11	10.9999	11
12	Attack Pattern 12	11.9989	12
13	Attack Pattern 13	12.9206	13
14	Attack Pattern 14	13.9877	14
15	Attack Pattern 15	14.9986	15
16	Attack Pattern 16	16.0000	16
17	Attack Pattern 17	16.6118	17
18	Attack Pattern 19	18.9989	19
19	Attack Pattern 20	19.9997	20
20	Attack Pattern 21	21.0000	21
21	Attack Pattern 22	22.0000	22
22	Attack Pattern 23	22.5392	23
23	Attack Pattern 24	23.9999	24
24	Attack Pattern 25	25.0000	25
25	Attack Pattern 26	25.9996	26
26	Attack Pattern 27	26.9935	27
Results from Network 2			
27	Attack Pattern 28	27.9970	28
28	Attack Pattern 29	28.9943	29
29	Attack Pattern 30	30.0000	30
30	Attack Pattern 31	31.0000	31
31	Attack Pattern 32	31.9999	32
32	Attack Pattern 33	32.6139	33
33	Attack Pattern 34	34.0000	34
34	Attack Pattern 35	34.9684	35
35	Attack Pattern 36	36.0000	36
36	Attack Pattern 37	37.0000	37
37	Attack Pattern 38	38.0000	38
38	Attack Pattern 39	39.0000	39
39	Attack Pattern 40	39.9916	40
40	Attack Pattern 41	41.5488	41
41	Attack Pattern 42	42.0000	42
42	Attack Pattern 43	43.0000	43
43	Attack Pattern 44	43.9998	44
44	Attack Pattern 45	44.9992	45
45	Attack Pattern 46	46.0000	46
46	Attack Pattern 47	47.0000	47
47	Attack Pattern 48	47.9992	48
48	Attack Pattern 49	49.0000	49
49	Attack Pattern 50	49.9751	50
50	Attack Pattern 51	50.8999	51
51	Attack Pattern 52	51.5942	52
52	Attack Pattern 53	52.6986	53

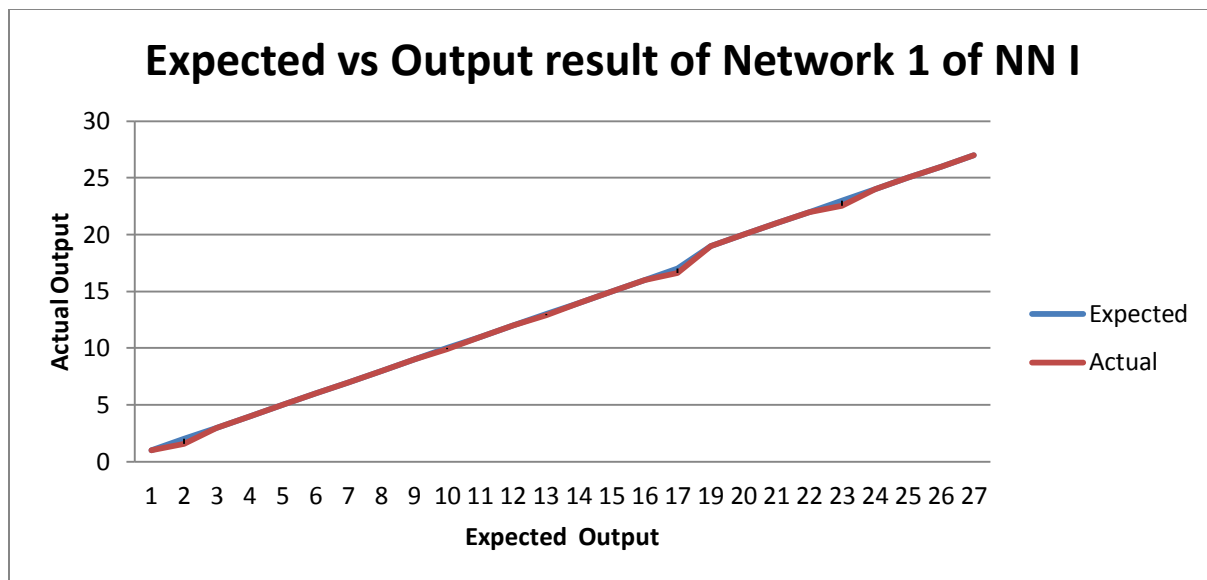


Figure 6. 1: Actual vs. Expected output of Network 1 of NN I

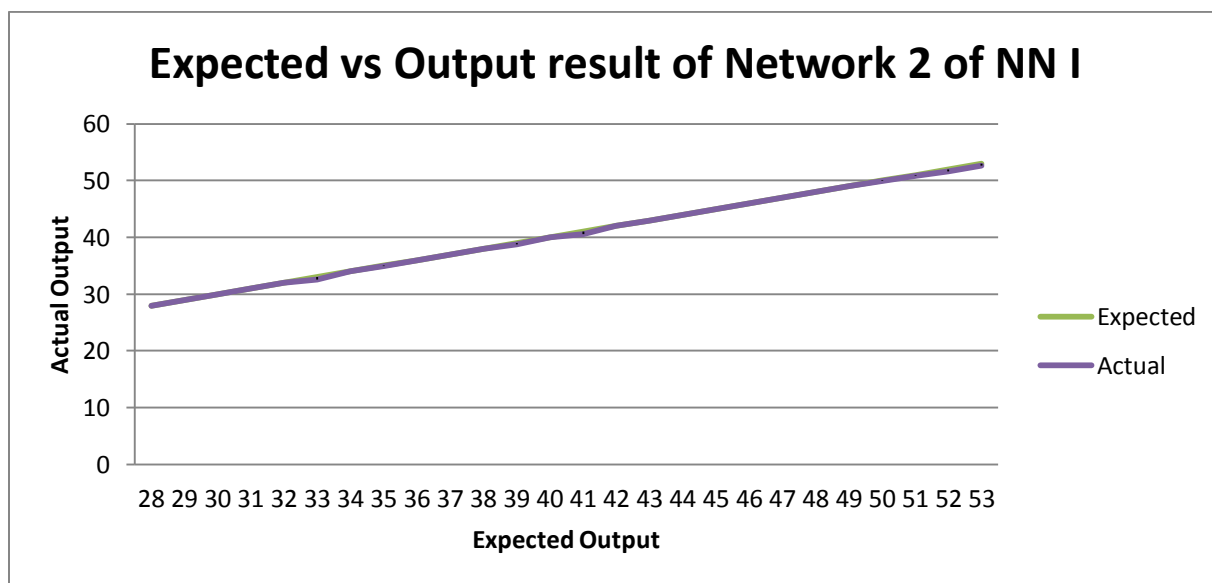


Figure 6. 2: Actual vs. Expected output of Network 2 of NN I

In comparing the actual output of neural network I to its expected output it would be seen that the neural networks have been able to match the correct attack patterns as close as possible in the expected output. Results from the actual output show that while the network was able to match some the attack pattern investigated exactly (i.e. identification of the exact attack pattern ID), the result also showed many attack patterns were identified by approximating their IDs. Of the 52 data sample that was used to test neural network I, 16 of the actual output result were exact match of expected output while the remaining 36 result were approximation of the attack pattern IDs. The approximated results are accepted in the validation study as this showed that the network was able to identify the possible attack pattern from the test data presented to it.

Analysis of the result of the actual output showed that the attack pattern that were exactly matched from the test data were attacks in which the attackers had manipulated the user input in order to stage buffer overflow attacks or denial of service attacks. This is no surprise as most of the attacks patterns involve the attacker manipulating the user input. However, for the outputs in which the network gave a result which matched the expected output after approximation (i.e. $-0.5 < X < 0.5$: where X is attack pattern ID), were mostly attacks causing DoS.

Michael Geigick and Laurie Williams conducted a feasibility study by using undergraduate students playing the role of software developers who are not experienced in security to match 20 of their attack patterns to the system design of a hypothetical banking system (see Appendix IV) using their SAFE-T approach. The result of the feasibility study showed that 91% of attack patterns were correctly identified by the students. Using design scenarios from the same system design, neural network I was also tested in a similar way to identify attack patterns that can be matched to the design scenarios. Table 6.8 shows the expected regularly expressed attack patterns for the system design and the actual output of the network for the attack patterns matched to each design scenario. The result showed that neural network I was able to match the possible attack pattern to each design scenario with an output which was close as much as the expected output. Comparing this result with the result of the neural network, it could be seen that the neural network gave a better performance by identifying all the attack patterns relating to the design correctly. Therefore, given the accuracy of the neural networks, it shows that neural networks can be used to evaluate software from its design

Table 6. 8: Actual and expected output of Neural Network I with input from design scenario

s\n	Design Scenarios	Actual Output	Expected Output
1	Scenario 1	0.9423	1
2	Scenario 2	1.9956	2
3	Scenario 3	2.9986	3
4	Scenario 4	3.9959	4
5	Scenario 5	4.9592	5
6	Scenario 6	5.7081	6
7	Scenario 7	6.9989	7
8	Scenario 8	7.9999	8
9	Scenario 9	8.8304	9
10	Scenario 10	9.9949	10
11	Scenario 11	10.9985	11
12	Scenario 12	11.9961	12
13	Scenario 13	12.9963	13
14	Scenario 14	13.9968	14
15	Scenario 15	14.9992	15
16	Scenario 16	16.9996	17
17	Scenario 17	19.9969	20
18	Scenario 18	20.9988	21
19	Scenario 19	22.5392	23

20	Scenario 20	23.9941	24
----	-------------	---------	----

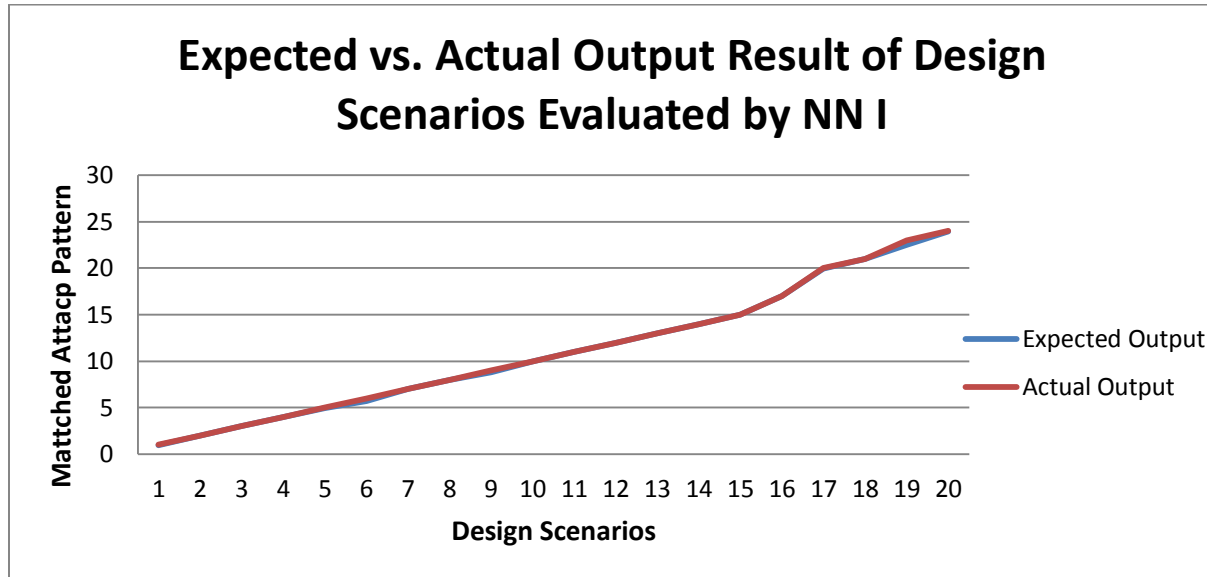


Figure 6. 3: Actual vs. Expected output result of design Scenarios evaluated by NNI

6.4.2 Neural Network II

The performance of neural network II was also tested using the test data in chapter 5 section 5. Table 6.9 shows the actual and expected output of neural network II. By comparing the expected and actual output, it would be seen that the network was able to match most the attack patterns to correct the group that will provide mitigation to vulnerabilities in the attack pattern. In a similar way to neural network I, the output of neural network II matched the expected output after approximation (i.e. $-0.5 < X < 0.5$: where X is group ID) There were two instances in which the network failed to match the attack patterns to the correct group. This was when the network was used to evaluate test data sample 10 and 17. For test data sample 10, the network produce an output of 2.6441 when the expected output is 6 and for test data 17, the network produced an output of 6 when the expected output is 2. By looking at the data used in training the network for matching the attack patterns to their corresponding security pattern, it was seen that for these attack patterns, the attacker had multiple ways in which the attack could be carried out. This explains why the network failed to match the attack patterns. With a larger data sample for training the neural network, a better performance can be achieved. Figure 6.4 is a graph showing the difference between the actual and expected output

Table 6. 9: Actual and expected output of Neural Network II

s\n	Test Data Sample	Actual Output	Expected Output
1	Sample 1	6.0000	6
2	Sample 2	5.9999	6

3	Sample 3	5.0000	5
4	Sample 4	4.9998	5
5	Sample 5	5.0000	5
6	Sample 6	5.0000	5
7	Sample 7	5.9999	6
8	Sample 8	6.0000	6
9	Sample 9	5.0000	5
10	Sample 10	2.6441	6
11	Sample 11	5.0000	5
12	Sample 12	5.0000	5
13	Sample 13	6.0000	6
14	Sample 14	4.9183	5
15	Sample 15	6.0000	6
16	Sample 16	5.0000	5
17	Sample 17	6.0000	2
18	Sample 18	1.7707	2
19	Sample 19	5.6890	6
20	Sample 20	4.0000	4

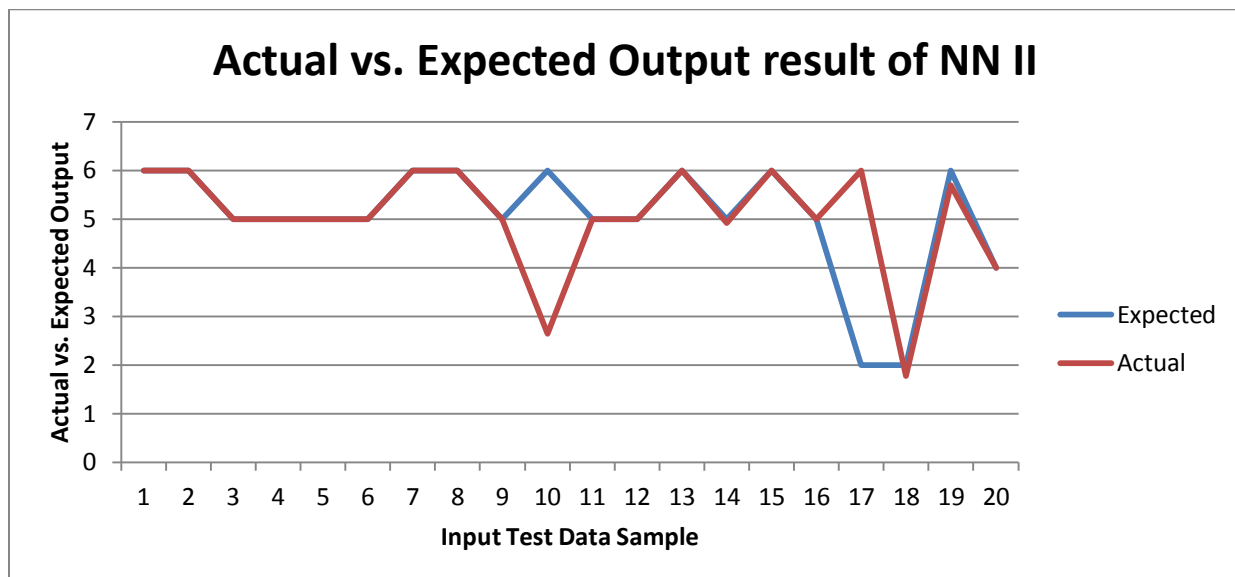


Figure 6. 4: Actual vs. Expected output of NN II

6.5. Case Study

The design for the online shopping portal by Mohan et.al (2009) is adopted in this research to illustrate how the proposed neural network tool can be used to evaluate software design for security flaws. In the design scenario, a customer visits the portal to either view or buy products. The customer searches for the product by selecting the appropriate category and brand. To purchase the product, the customer adds the product to the shopping cart which he can also edit by deleting already added product and adding new product. Once the customer

has completed his shopping, he checks out and he is prompted to sign in if he is an existing customer or to register if he is not. After signing in, customer is directed to the secure payment section where he confirms the delivery address, and also provides his payment details. At the last stage of the order, a confirmation page is displayed to confirm shipment ID and delivery of product with 15 days. From the design, three scenarios were evaluated. The identified attack patterns for each scenario by neural network I and the group of security patterns that can provide mitigation identified by neural network II is presented below. Figure 6.5 is the class diagram for the design of the online shopping portal.

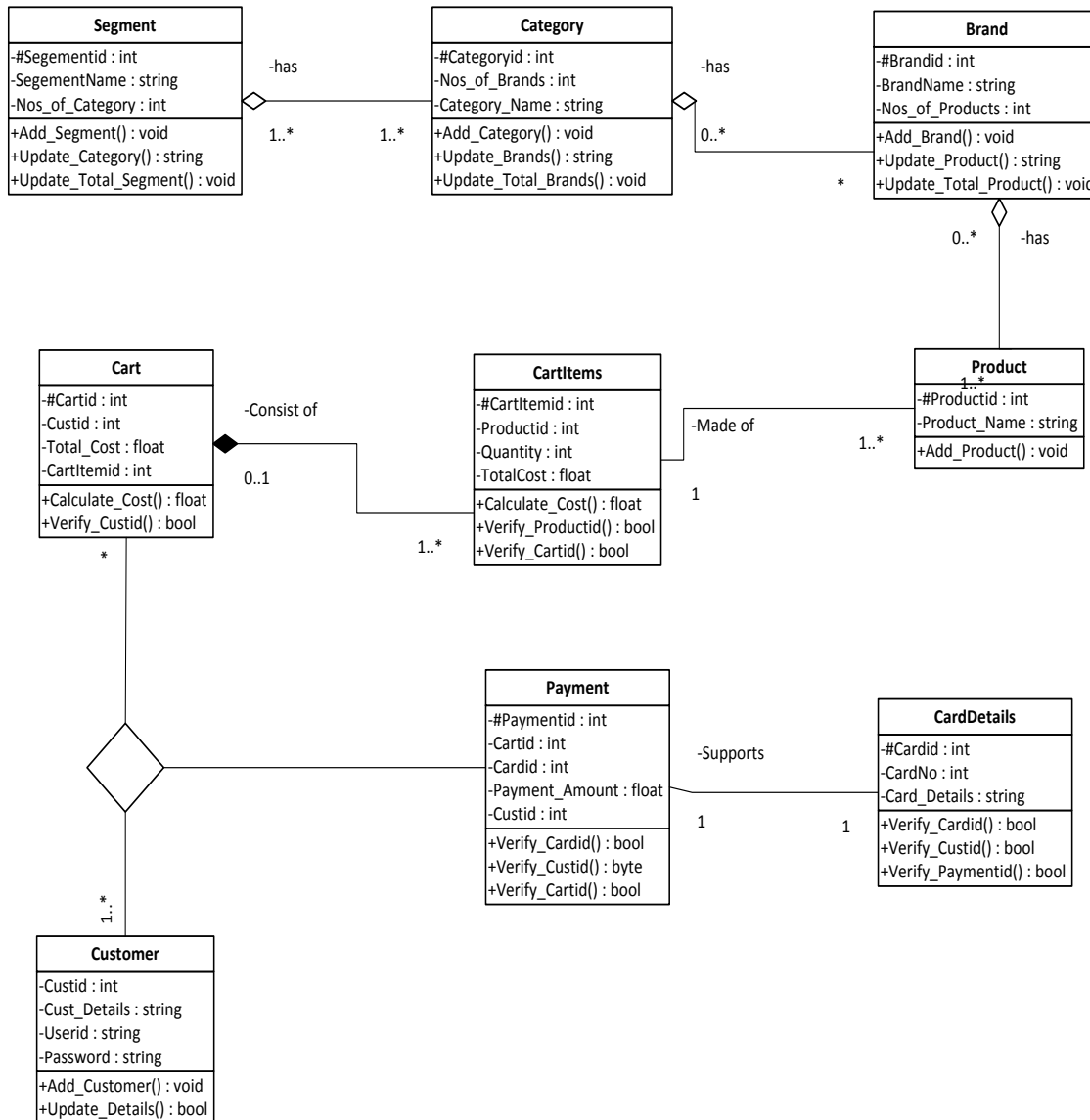


Figure 6. 5: Class diagram of online shopping portal

6.5.1 Scenario 1: Product Selection

In this design scenario, the customer visiting the portal searches the portal by selecting the product category and then the brand to view different products. Product selected by the

customer is described as the cart item when it is added to the cart in the class diagram in Figure 6.5. The system returns a cart id for each cart item added to the cart. Figure 6.6 is the sequence diagram for this scenario. Looking at the scenario, information stored by the cart could be chosen by the attacker as the target of attack. By manipulating this information, it is possible for the attacker to reduce the price to be paid for the products that have been added to the cart. Figure 6.7 shows that the data is stored in a backend database. Based on the attack attributes in Table 4.1 in chapter 4 section 3, the data capturing the design scenario on Table 6.10 is generated.

Table 6. 10: Attack attributes for scenario 1

S\n	Attributes	Observable	value
1	Attacker	No access	0
2	Source of Attack	External	1
3	Target of Attack	Information	54
4	Attack Vector	Query String	77
5	Attack Type	Confidentiality	9
6	Input validation	No Validation	3
7	Dependencies	Input validation and access control	6
8	Output encoding	Escaping supplied user input (lacking)	3
9	Authentication	None	0
10	Access Control	Function access	4
11	HTTP Security	Input validation	3
12	Error handling and Logging	None	0

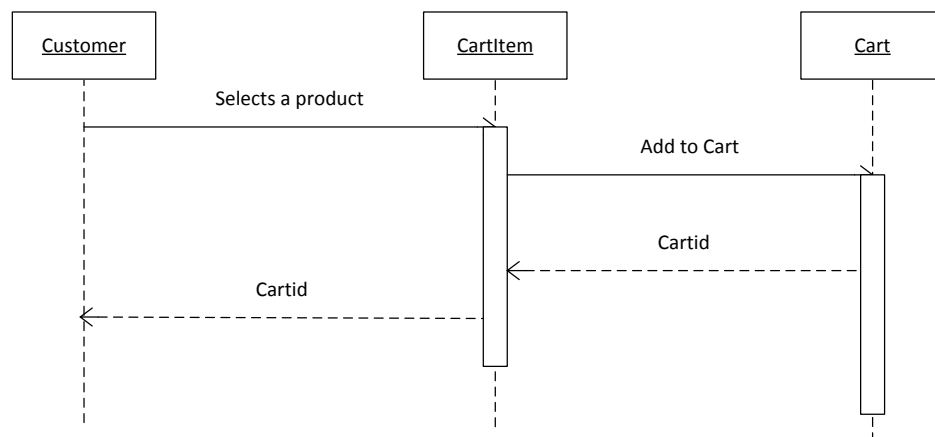


Figure 6. 6: Sequence diagram for product selection

By using the corresponding values of the attributes in Table 6.4 as the input for neural network I, the network produced an output of 25.9976 which corresponds to regularly expressed attack pattern 26. In this attack, an attacker can use directory traversal or shell characters as input in a crafted URL in order to view sensitive information (See Table 3.2 in Chapter 3 section 5 for more discussion). An example of this attack is recorded by security focus under BID 3308

and in CVE details database under CVE 2001-0985. In order to match the identified attack pattern to the corresponding group of security patterns that can provide mitigation, the attributes of the attack patterns listed in Table 5.5 in chapter 5 section 3 is used to generate the input for neural network II (See Table 6.11). With this input, neural network II produced an output of 3.9615. This corresponds to group 4 of the security patterns in Table 5.15 in chapter 5 section 3. This group of security pattern provides mitigation for information disclosure attacks.

Table 6. 11:Attributes of identified attack pattern in scenario 1

S\n	Attributes	Observable	Value
1	Attack ID	Attack ID	26
2	Resource Attacked	Information	54
3	Attack Vector	Query String	77
4	Attack type	Confidentiality	3

6.5.2 Scenario 2: Cart Submission

In this scenario, the customer checks out after adding products he decides to buy to the shopping cart. The cart checks out each cart item by requesting the cart item class to calculate the cost. Following this, data on the cart is stored on the database which generates a cart id for the cart. In this design scenario, the attacker can choose the data stored in database as his target of attack by manipulating the data on the database. Figure 6.7 shows the sequence diagram of this scenario. The data capturing the design scenario using the attack attributes is shown in Table 6.12

Table 6. 12:Attributes for scenario 2

S\n	Attributes	Observable	Value
1	Attacker	No Access	0
2	Source of Attack	External	1
3	Target of Attack	Data	20
4	Attack Vector	SQL Input	90
5	Attack Type	Privacy	9
6	Input validation	No validation	3
7	Dependencies	Validation and access Control	6
8	Output encoding	None	0
9	Authentication	None	0
10	Access Control	Service access	3
11	HTTP Security	Input Validation	3
12	Error handling and Logging	None	0

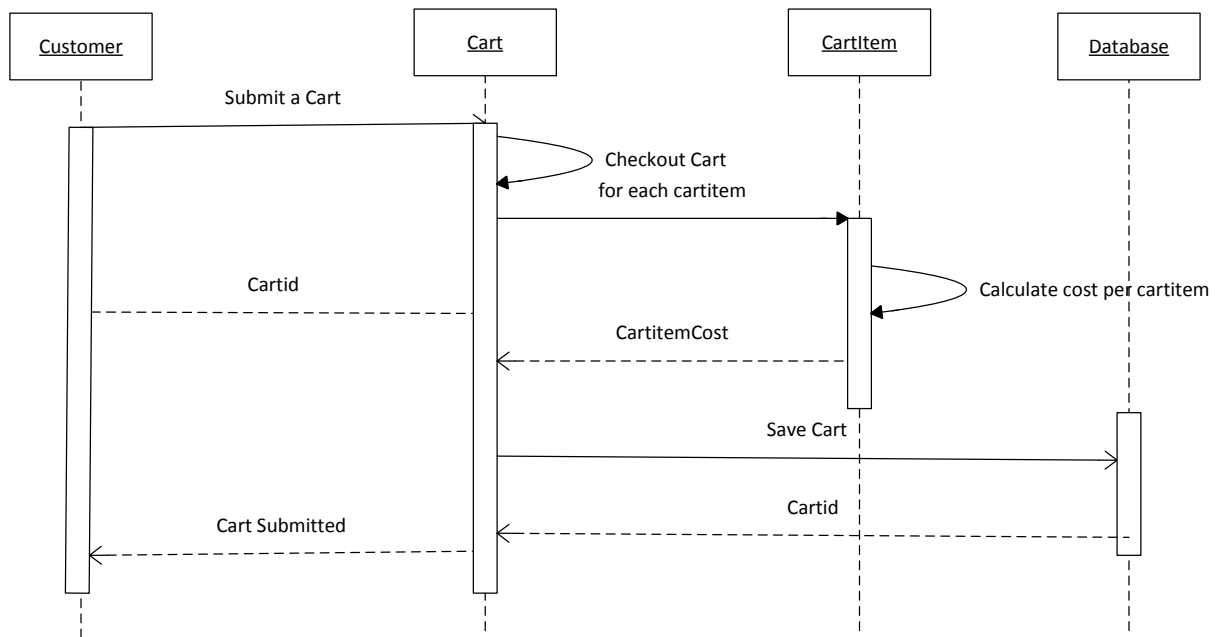


Figure 6. 7: Sequence diagram for shopping cart submission

Neural network I produced an output of 8.9988 when the value of the attributes on Table 6.12 was used as its input. This corresponds to the regularly attack pattern 9 in which the attacker injects SQL into the URL in order to query backend databases. An example of this attack is recorded by security focus under BID 9967 and in CVE details database under CVE 2004-2412. Table 6.13 shows the attributes for the identified attack pattern and their corresponding values which are used as input for neural network II. The network produced an output of 1.9999 which indicates that the security design patterns in group 2 as shown on Table 5.13 in chapter 5 section 3 can provide mitigation for data tampering attacks.

Table 6. 13:Attributes of identified attack pattern in scenario 2

S\n	Attributes	Observable	Value
1	Attack ID	Attack ID	9
2	Resource Attacked	Data	20
3	Attack Vector	SQL Input	91
4	Attack type	Integrity	2

6.5.3 Scenario 3: Log in

In the log in design scenario, the customer is prompted to log into his account by supplying his log in credentials. The log credentials are then validated by checking the details of the user on the database. If the credentials supplied by the user are valid, the user is granted access to his account. Figure 6.8 shows the sequence diagram for this scenario. A point of interest for an attacker in this particular design scenario is gaining access into various accounts in the shopping portal. The data capturing the design scenario using the attack attributes is shown in Table 6.14

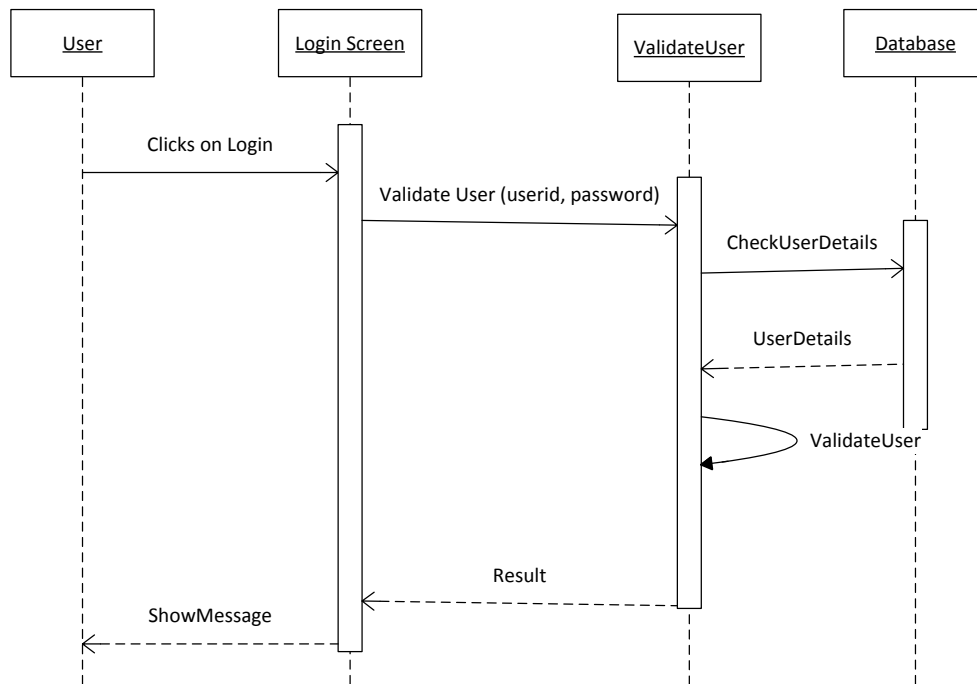


Figure 6. 8: Sequence diagram for customer login

Table 6. 14:Attributes for scenario 3

S\n	Attributes	Observable	Value
1	Attacker	No access	0
2	Source of Attack	External	1
3	Target of Attack	Authentication Routine	5
4	Attack Vector	Username Entry	101
5	Attack Type	Confidentiality	9
6	Input validation	No Validation	3
7	Dependencies	Validation	3
8	Output encoding	None	0
9	Authentication	None	0
10	Access Control	Service access	3
11	HTTP Security	Input Validation	3
12	Error handling and Logging	None	0

Using the values on Table 6.14 as input for neural network I, the network produced an output of 7.9786 which corresponds to regularly expressed attack pattern 8. In this attack, the attacker submits a long string of character for the username. This causes a buffer overflow that enables the attacker to escalate his privileges. An example of this attack is recorded by security focus under BID 9672 and in CVE details database under CVE 2004-0286. Table 6.15 shows the attributes for the identified attack pattern and their corresponding values which are used as

input for neural network II. The network produced an output of 6.0000 which indicates that the security design patterns in group 6 as shown on Table 5.17 in chapter 5 section 3 can provide mitigation for privilege escalation attacks.

Table 6. 15: Attributes of identified attack pattern in scenario 3

S\n	Attributes	Observable	Value
1	Attack ID	Attack ID	8
2	Resource Attacked	Authentication Routine	5
3	Attack Vector	Username Entry	101
4	Attack type	Confidentiality	3

6.6 Comparison of the neural network approach with current approaches

In comparing the proposed neural network tool with the current approaches used in integrating security into software design during SDLC, the following can be observed:

In a similar way to architectural risk analysis and threat modelling, it is necessary to decompose the software architecture so as to identify the features of the software design (such as assets and source of attack) that needs to be analysed when using the proposed neural network tool. For example, in the case study in above, each of the design scenarios represent a functional part of the software design which is analysed using the attack attributes to abstract the data needed by the neural network for analysing the software design.

Furthermore, the proposed neural network tool is based on the abstract and match technique through which software flaws in a software design can be identified when an attack pattern is matched to the design. Hence, using well known approaches such as DFD and sequence diagrams familiar to software developers, they are able to abstract information about their software designs needed by the Neural Network tool for matching possible attack patterns (Adebiyi et.al, 2012).

However, to conduct architectural risk analysis and threat modelling may be daunting task for software developers who are not necessarily experts in security. Therefore, the need to involve security experts to analyse the software design for security flaws is inevitable. The proposed neural overcome this challenge by helping software developers to think of the defences to be put in place when possible attack patterns are matched to their software design.

Also, as discussed in chapter 2 section 11, current security tools used during software design such as SDL Threat Modelling Tool is limited by the knowledge of the software developer creating the threat Model. As the proposed neural network tool aids software developers who are non-security experts, this problem is addressed as the developer only needs to abstract the information needed from their designs to be analysed. This eliminates the need for software developers to think like the attacker when conducting threat modelling or when drawing attack

trees. All the developers need to focus on is the software design and abstracting the correct information from the design scenarios. Moreover, the interpretation of results generated from current security tools depends largely on the understanding of the developer on security risks. In contrast, the attack pattern matched to a software design and the security design patterns identified as solution to flaws in the software design using the proposed neural network tool can be easily understood by software developers.

While the use of formal methods helps to eliminate software flaws in software during SDLC, as discussed in chapter 2 section 8, it is not widely used by software developers. Apart from this its adoption by software developers may require a significant deviation from their software development methodology. However, the proposed neural network tool can be used during the design phase of current software development methodologies without a significant deviation from software development process.

6.7. Summary of Chapter 6

The statistical analysis conducted on the performance of the networks in this chapter showed that there were significant differences in their performance when SCG and RP training optimization algorithms were applied to the networks. From the results on the performance of the neural networks in the validation study, it was observed that that neural network I was able to match the test data sample to the expected attack patterns as close as possible. Neural network II was also able match most of the test data sample to the expected group of security design patterns. In the two instances the network failed in identifying the correct group, it was observed that attack patterns had multiple ways in which the attack could be carried out. This explains why the network has not been to match the test data sample for these attack patterns to the correct group. In the case study, three design scenarios from the online shopping portal were evaluated. Using the attack attributes, data needed from the scenarios were abstracted and were used as input for the neural network. The result of the evaluation show that the neural network I was able to match possible attack patterns to the design of the shopping portal and neural network II was able to match the identified attack patterns to the group of security design patterns that can provide mitigation for the attacks. Based on this information, a software developer can make informed decision on what to do in order to integrity security into his software design. A comparison between the proposed neural network approach and current approaches was also discussed in this chapter. In chapter 7, the conclusion to this research is presented and the future direction is discussed.

Chapter 7. Conclusion

Securing software products no doubt will continue to be an on-going challenge because of the nature of software produced today. As consumers continue to demand the addition of new features to various software applications and the software developers aim at meeting the time to market the software applications, malicious hackers will continue to explore new forms of vulnerability which they can exploit. As a result, there is no time when new forms of attacks and vulnerabilities will not be discovered. It must be noted that network security cannot guarantee the security of software applications within a network because software base attacks are designed to follow the normal path of a software functionality which already provides an acceptable means of access through a network and its security control.

It cannot be overstated that the cost of fixing security flaws in software applications is very costly after they are deployed. The cost could be 30 times more than the cost of finding and fixing the problem early in the SDLC. Therefore, integrating security into a software design will help tremendously in saving time and money during software development and when the software is deployed. For instance, it is less expensive and less disruptive to discover design-level vulnerabilities during the design, than during implementation or testing, forcing a costly redesign of pieces of the application.

Therefore, software developers must begin to work towards building more secured software products by making security a top priority during SDLC. In this regard, software development team should be aware of the security issues affecting the software under development and must be properly trained to ensure that these issues are properly addressed during SDLC. The use of tools capable of helping developers to do a better job and software development process that integrates security throughout SDLC also plays a very important role in producing secure software. Base on this fact, this research work proposed the use of neural network as a tool to enable software developers to evaluate their software design for security flaws and also suggest possible solutions

Chapter 1 gives the background to this research work and in chapter 2 a literature review on the current approaches used in securing software applications was conducted. Based on the information gathered on two of the current approaches, the neural network tool in this research was proposed. Chapter 3 discussed the proposed neural network tool while chapter 4 and 5 demonstrated the implementation of the first and second neural network respectively. Chapter 6 presented the results and discussion on the performance of the neural network. In this chapter, the discussion on the contribution on this research work to knowledge, the limitation of the proposed neural network model and a comparison of the proposed neural network approach with some of the current approaches used in integrating security into

software design are presented. This chapter concludes with the future direction of this research work.

7.1 Contribution to Knowledge

The major contributions of this research work to knowledge are as follows:

1. Matching attack patterns to software design. By using of neural network as a tool for matching attack patterns to software design during design phase of SDLC, the security flaws in the software design can be identified. The identification of the security flaw in the software design by the neural network tool will enable software developers to take the necessary steps in mitigating the threat identified in the security flaw before coding begins.
2. Matching attack patterns to security design patterns. The use of security design pattern to resolve security problems is currently a challenge to software developers. Building on the existing approach in literature to match security design patterns to attack patterns, the proposed neural network tool matches identified attack patterns in a software design to the corresponding security patterns that can provide mitigation.
3. Aids towards bridging the gap between software developers and security professionals. When attack patterns are matched against software design by the neural network tool, software developers become more aware on the security aspect of their software design and security expertise solutions to threat in the attack. In this way, the software developers can benefit from security expertise of the security professionals.

Therefore, based on the above, the success of this research in using neural networks to evaluate software design for security flaws will consolidate the efforts of software designers evaluating their software as they identify areas of security weakness in their software design. This will enhance the development of secured software applications in the software industry especially as software designers often lack the required security expertise. Thus, neural networks given the right information for its training will also contribute in equipping software developers to develop software more securely especially in the area of software design.

7.2 Limitation

One of the significant limitations to this research is the difficulty of obtaining information from software developers in the industry on software designs that can be used in this research work. As this information is private to the software developers, limited information was obtained. Another difficulty encountered during the course of this research work is the representation of the software design to neural network. As there is no previous work related to using neural network as a tool during software design stage of SDLC, no information could be obtained from

previous research papers. However, after looking through the resources on neural network repository online and speaking to some software developers in various conferences attended, this problem was resolved. One of the valuable lesson learnt during this research work, is to constantly seek professional training relevant to my area of research as this contributes significantly to the research work. For instance, it was after going some series of training on Mat lab that the skill needed for training the neural network in this research work was acquired.

The regularly expressed attack pattern used in training the neural network is a generic classification of attack patterns. Therefore, any unknown attack introduced to the neural network will be classified to the closet regularly expressed attack pattern. However, the success of the neural network in evaluating software design for security flaws largely depends on the input data capturing the attributes of the software design introduced to it.

7.3 Future Work

Various security tools have been developed to aid software developers in integrating security into software applications during SDLC. However, most of these tools are used in the late phase of SDLC. As the focus of this research work is integrating security during the early phase of SDLC especially during the design phase, it is our intention to carry out a comparative study in which the neural network tool would be compared to security tools currently used in integrating security during the design phase of SDLC. Result of this comparative analysis should include how effective it is in identifying security flaws and its performance based on security background of the user using the tools.

Furthermore, information from attack patterns which capture security flaws in software designs from other authors and from CAPEC (Common Attack Pattern Enumeration Classification) would be used in training the neural network. This would subsequently improve the performance of the network and increase its scope in matching design scenarios to attack patterns not covered in the regularly expressed attack patterns. Also, the neural network tool needs to be trained to match attack patterns to other security design patterns proposed by other authors. It is also our intention to look further into improving the output result of the neural network tool by using other suggested classification of security patterns when defining the expected output of the neural network II.

Further testing of the neural network tool is also required before it can gain acceptance as a tool for matching attack patterns to software designs and matching attack patterns to security patterns. This test would include finding out whether the attack described in an attack pattern matched to a design by the neural network tool is feasible. And if this is feasible, another test

should be conducted to explore how effective is the solution that has been suggested in preventing the attack

For the neural network tool to be used by developers, it could be deployed as a plugin in an existing software design suite E.g. IBM Rational Rose. The neural network tool will work within the same design suite and provide feedback to developers. Also, it would be of great benefit if the tool could connect to online vulnerability database and CAPEC to help developers get detailed information on threat that has been identified. Alternatively, it could be deployed as an independent security tool. However, for the neural network tool used successfully, it must be able to read files created from software design suite and must be able to synchronize its feedback to such tools. This is because most software design suite encourage collaboration among all the stakeholders in a software project and feedback from the neural network tool needs to be accessible to the relevant stakeholders

Lastly, it is desirable to also find out the impact of the neural network tool on the training and security awareness of software developers and on other security techniques used in integrating security in software during software development lifecycle. This could be investigated by measuring how the feedbacks from the neural network tool add to the knowledge of software developers on the security of their software application and how it complements other security techniques.

References

- Adebiyi, A. et.al., (2012), 'Evaluation of Software Design using Neural Network', In the proceedings of the 8th International Conference on Web Information Systems and Technologies (WEBIST), Porto, Portugal
- Adebiyi, A. et.al., (2012), 'Matching Attack pattern to Security Pattern using Neural Network', Paper accepted for the European Conference on Information Warfare and Security (ECIW-2012), Paris, France
- Adebiyi, A., Lee S.W, Mouratidis, H. and Imafidon C.,(2012), 'Applicability of Neural Network to Software Security'. Abstract accepted at the ACT 2012 conference, London, United Kingdom
- Adebiyi A., et.al., (2012), 'Applicability of Neural Network to Software Security' In proceedings of the UKSim 14th International Conference on Computer Modelling and Simulation ,Cambridge, United Kingdom, pp19-24
- Amey, P. (2006) Correctness by Construction, Build Security In, Available at: <https://buildsecurityin.us-cert.gov/bsi/articles/knowledge/sdlc/613-BSI.html>(Last Accessed: December 2011)
- Agarwal, A. (2006), 'How to Integrate Security into your SDLC',*SearchAppSecurity.com*, Available at: http://searchsoftwarequality.techtarget.com/tip/0,289483,sid92_gci1174897,00.html (Last Accessed: October 2011)
- Ahmad, I. Swati, S.U. and Mohsin, S. (2007), 'Intrusion Detection Mechanism by Resilient Back Propagation (RPROP)', *European Journal of Scientific Research*, Vol. 17(4), pp523-530
- Barnum, S. and Gegick, M. (2005) 'Design Principles, Build Security In', Available at: <https://buildsecurityin.us-cert.gov/bsi/articles/knowledge/principles/358-BSI.html> (Last Accessed: December 2011)
- Beale, M. H., Hagan, M. T. and Demuth, H. B. (2010), 'Neural Network Toolbox 7 User Guide, Mathworks', Available at: http://www.mathworks.com/help/pdf_doc/nnet/nnet.pdf. (Last Accessed March 2012)
- Berg, B. (2010), 'SDL: Threat Modelling tools vs. Threat Analysis tool', Available at: <http://www.dib0.nl/code/166-sdl-threat-modeling-tool-vs-threat-analysis-tool> (Last Accessed: November 2011)
- Berg, R. (2007), 'Secure at the Source: Implementing Source Code Vulnerability Testing in the Software Development Lifecycle' *Ounce Lab Inc.*, Waltham, MA. Available at: http://www.paramountassure.com/pdf/Source_code_vulnerability_testing_in_SDL.pdf (Last Accessed March 2013)
- Bivens et.al. (2002), 'Network Based Intrusion Detection Using Neural Network', In *Proceedings of Artificial Neural Networks In Engineering (ANNIE) Conference 2002*, St. Louis, Missouri pp10-13

Blackley, B. et.al (2004) 'Technical Guide Security Design Patterns', The *Open Group*, Available at: http://users.uom.gr/~achat/articles/sec_patterns.pdf (Last Accessed: December 2011)

Bunke, M., et.al, (2011) 'Application-Domain Classification of Security Patterns, In: *The Third International Conferences on Pervasive Patterns and Applications*, Rome, Italy, pp 138-143

Cannady, J. (1998), 'Artificial Neural Networks for Misuse Detection', In *Proceedings of the 21st National Information Systems Security Conference*, Virginia, USA, pp368-381

Connolly, T. and Begg, C. (2005), 'Database Systems : A Practical Approach to Design, Implementation and Management', 4th Ed, Addison-Wesley, USA

Chickowski, E. (2010), 'Lessons Learned from Five Big Database Breaches in 2010', *Dark Reading*, Available at: <http://www.darkreading.com/database-security/167901020/security/attacks-breaches/228900094/lessons-learned-from-five-big-database-breaches-in-2010.html> (Last Accessed: December 2011)

Croxford, M. (2005), 'The challenge of low defect, secure software- too difficult and too expensive', *Secure Software Engineering*, Available at: <http://journal.thedacs.com/issue/2/33> (Last Accessed: February, 2012)

Croxford, M and Chapman, R. (2005), 'Correctness by Construction: A Manifesto for High Software', *The Journal of Defence Software Engineering*, Available at: <http://www.crosstalkonline.org/storage/issue-archives/2005/200512/200512-Croxford.pdf>

CWE (2013), 'CWE-285: Improper Authorization', *Common Weakness Enumeration*, Available at: <http://cwe.mitre.org/data/definitions/285.html>, (Last Accessed: March 2012)

Davis, N. (2005), 'Developing Secure Software', *Secure Software Engineering*, Available at: <http://softwaretechnews.com/stn8-2/noopur.php> (Last Accessed: November 2011)

Dong J. et al (2009), 'Automated verification of security pattern compositions', *Information and Software Technology*, Vol. 52 (3), p274- p29

Edward, R. J. (2008), 'Neural Networks' Role in Predictive Analytics' *Information Management*, Available at: http://www.information-management.com/specialreports/2008_61/-10000704-1.html (Last Accessed March 2013)

Erbshloe, M. (2002), 'Economic Impact of Network Security Threats', *Cisco Systems Inc*, Available at: http://www.cisco.com/warp/public/cc/so/neso/sqso/roi1_wp.pdf (Last Accessed: March 2011)

Fernandez, E.B et.al, (2007) 'Security patterns for Voice over IP' *Journal of Software*, Vol.2(2), 19-29.

Gavin, H. (2011), 'The Levenberg-Marquardt method for nonlinear least squares curve-fitting problems' Available at: <http://www.duke.edu/~hpgavin/ce281/lm.pdf> (Last Accessed: March 2012)

Gegick, M. and Williams, L. (2007), 'On the design of more secure software-intensive systems by use of attack patterns', *Information and Software Technology*, Vol. 49, pp 381-397

Gregory L. S.,(2011), 'Glossary', *Texas State Library and Archive Commission*, Available at: <https://www.tsl.state.tx.us/ld/pubs/compsecurity/glossary.html>, (Last Accessed: March 2012)

Greenberg, A.(2008), 'A tax on buggy software', *Forbes.com*, Available at: http://www.forbes.com/technology/2008/06/26/rice-cyber-security-tech-security-cx_ag_0626rice.html (Last Accessed: August 2011)

Goertzel, K.M. et.al.(2006) 'Security in the Software Life Cycle: Making Software Development Processes—and the Software Produced by Them—More Secure', Draft Version 1.2, *Department of Homeland Security (DHS)*, US

Hafiz, M. and Johnson, E.(2006) 'Security Patterns and their Classification Schemes' *Technical Report for Microsoft's Patterns and Practices Group*, Available at: <http://munawarhafiz.com/research/patterns/secpatclassify.pdf> (Last Accessed: December, 2011)

Hall, A and Chapman, R, (2004), 'Software Engineering, Correctness by Construction', Available at: http://www.anthonysmall.org/Correctness_by_Construction.pdf

Halkidis, S.T. et.al (2006), 'A qualitative analysis of Software security patterns', *Computer & Security*, Vol. 25, p379-p392

Hinchey, M et al, (2008), 'Software Engineering and Formal Methods', *Communications of the ACM*, Vol.51(9), pp54-59

Hoglund, G.(2002), 'Bad Software', *Cenzic, Inc*, Available at: <http://www.blackhat.com/presentations/bh-usa-02/bh-us-02-hoglund-software.ppt> (Last Accessed: January, 2011)

Hoglund, G and McGraw G. (2004), 'Exploiting Software: The Achilles' Heel of CyberDefense', *CyberDefense Magazine*, Available at: http://citigal.com/papers/download/cd-Exploiting_Software.pdf (Last Accessed: June 2011)

Hoglund, G and McGraw G. (2004), 'Exploiting Software: How to Break Code', *Addison Wesley*, Boston, USA

Hoglund, G. and McGraw G (2007), 'Online Games and Security', *IEEE Computer Society*, Available at: <http://www.cigital.com/papers/download/attack-trends-EOG.pdf> (Last Accessed: January 2011)

Howe (2005), 'Crisis, What Crisis?', *IEEE Review*, Vol. 51(2), p39

Hope, P.; Lavenhar, S. and Peterson, G.(2008), 'Architectural Risk Analysis', *The Build Security In (BSI) Portal*, Version 28, Available at: <https://buildsecurityin.us-cert.gov/daisy/bsi/articles/best-practices/architecture/10-BSI.html> (Last Accessed: August 2011)

Ho, S. L.; Xie, M. and Goh, T. N. (2003), 'A Study of the Connectionist Model for Software Reliability' *Computer and Mathematics with Applications*, Vol. 46, 1037 -1045

Humphrey W. S.(2004), 'Defective Software Works, Carnegie Mellon', Software Engineering Institute, Available at: http://www.sei.cmu.edu/news-at-sei/columns/watts_new/2004/1/watts-new-2004-1.htm (Last Accessed: December 2011)

Hu, V.C., Ferraiolo, D. F. and Kuhn, D. R. (2006), 'Assessment of Access Control Systems' *National Institute of Standards and Technology*' Available at: <http://csrc.nist.gov/publications/nistir/7316/NISTIR-7316.pdf> (Last Accessed: February 2013)

Jackson, D, (2006), 'Dependable Software System by Design', *Scientific American, Inc.*, Available at: http://www.cs.virginia.edu/~robins/Dependable_Software_by_Design.pdf

Jaspreet (2012), 'Security Breaches are on the Increase but Preventable', *Druva*, Available at: <http://www.druva.com/blog/2012/08/15/security-breaches-are-on-the-rise-but-preventable/> (Last Accessed: April 2013)

James, D. (2011), 'Top 10 Information Security Breaches', *Ascentor*, Available at: <http://www.ascentor.co.uk/2011/10/top-10-information-security-breaches/> (Last Accessed: April 2013)

Jones, A. (2010), Better: Invalidation or Output Encoding? Available at: <http://msmvps.com/blogs/alunj/archive/2010/05/31/1771098.aspx> (Last Accessed: November 2011)

Joseph, A., Bong, D.B.L. and Mat, D. A. A.(2009), 'Application of Neural Network in user Authentication for Smart Home Systems', *World Academy science, Engineering and Technology*, Vol. 53, pp1293-1300

Karras, D.A. and Zorkadis, V. (2003), 'On neural network techniques in the secure management of communication systems through improving and quality assessing pseudorandom stream generators', *Neural Networks*, Vol. 16, pp899 – 905.

Kenneth R. Van, W. and McGraw, G. (2006), 'Bridging the Gap between Software Development and Information Security', *IEEE Computer Society*, Available at: <http://www.cigital.com/papers/download/bsi10-ops.pdf> (Last Accessed: December 2011)

Kienzle, D. M and Elder, M. C. (2002) 'Final Technical Report: Security Patterns for Web Application Development', Available at <http://www.scripts.net/~celser/securitypatterns/final%20report.pdf>, (Last Accessed: January 2012)

Kiiski, L (2007) 'Security Patterns in Web Applications', *Publications in Telecommunications Software and Multimedia Laboratory*, Available at: http://www.tml.tkk.fi/Publications/C/25/papers/Kiiski_final.pdf (Last Accessed: November 2011)

Kim T. et al., (2000), 'Software Architecture Analysis: A Dynamic Slicing Approach', *ACIS International Journal of Computer & Information Science*, Vol. 1 (2), pp91-p103

Koskinen, J.(2003), 'Software Maintenance Costs', Information Technology Research Institute, Available at: <http://users.jyu.fi/~koskinen/smcosts.htm> (Last Accessed: January 2012)

Laverdiere M.A. et.al (2006), 'Security Design Patterns: Survey and Evaluation', IEEE CCECE/CCGEI, Ottawa

Lian, S. (2008), 'A block cipher based on chaotic neural networks', *Neurocomputing*, doi:10.1016/j.neucom.2008.11.005 (Last Accessed: January 2012)

Lin. I.; Ou, H. and Hwang, M (2005), 'A user authentication system using back-propagation network', *Neural Computer and Application*, Vol. 14, 243-249

Lourakis, I.A. (2005), 'A brief Description of the Levenberg Marquardt Algorithm Implemened by levmar', *Matrix*, Vol. 3, p2, Available at: <http://www.ics.forth.gr/~lourakis/levmar/levmar.pdf> (Last Accessed: March 2011)

Malaiya Y. K, et al. (1992), 'Using Neutal Networks in Reliability prediction', *IEEE Software*, pp53-59
Mahmoud, Q. (2000) 'Security Policy: A Design Pattern for Mobile Java Code' In: *Proceedings of the Seventh Conference on Pattern Languages of Programming (PLoP' 00)*, Illinois, USA.

MathWorks, (2011) Neural Network Toolbox, Available at: <http://www.mathworks.com/products/neuralnet/description3.html> (Last Accessed: January 2012)

McAvinney, A. and Turner, B. (2005), 'Building a Neural Network for Misuse Detection', *Proceedings of the Class of 2006 Senior Conference*, pp27-33

McGraw, G and Viega, J. (2001), 'Introduction to Software Security', *Inform IT Network*, Available at: <https://www.informit.com/articles/article.aspx?p=23950&seqNum=11> (Last Accessed: January 2012)

McGraw, G. (2002), 'Building Secure Software: Better that Protecting Bad Software', *IEEE Software*, Vol. 19(6), p57.

McGraw, G. (2003), 'Building Secure Software: A difficult but critical step in protecting your business', *Cigital, Inc*, Available at: http://www.cigital.com/whitepapers/dl/Building_Secure_Software.pdf(Last Accessed: November 2011)

McGraw, G.(2004), '*Software Security*', The IEEE Computer Society, pp80-83

McGraw, G. (2004), 'Who should do Security? Network Magazine', Vol.19(19), p72.

McGraw, G. (2006)'The Role of Architectural Risk in Software', *Inform IT Network*, Available at: <http://www.informit.com/articles/article.aspx?p=446451>(Last Accessed: November 2011)

McGraw, G.(2008), 'Software (In)security: Paying for Secure Software', *Inform IT Network*, Available at: <http://www.informit.com/articles/article.aspx?p=1189519> (Last Accessed: August 2011)

McGraw, G. (2008), 'Software (In)security: Securing Web 3.0', *Inform IT Network*, Available at: <http://www.informit.com/articles/article.aspx?p=1217101> (Last Accessed: August 2011)

Meier, J. D.; Mackman, A. and Wastell, B. (2005), 'Threat Modelling Web Applications', *Microsoft Corporation*, Available at: <http://msdn.microsoft.com/en-us/library/ms978516.aspx> (Last Accessed: October 2011)

Miller, M. (2008), 'Modelling the Trust Boundaries Created by Securable Objects ', In: *Proceedings of the 2nd USENIX Workshop on Offensive Technologies*, San Jose, CA.

Mockel C and Abdallah, A.E (2011) 'Threat Modelling Approaches and Tools for Securing Architectural Designs of E-Banking Application', *Journal of Information Assurance and Security*, Vol. 6(5), pp 346-356

Mouratidis, H. and Giorgini, P. and Schumacher, M. (2003), 'Security Patterns for Agent System', In: *Proceedings of the 8th European Conference on Pattern Languages of Programs 2003*, Irsee- Germany, ppC4-1 –C4-16

Mouratidis, H. and Giorgini, P (2007), 'Security Attack Testing (SAT)- testing the security of information systems at design time', *Information Systems*, Vol. 32, p1166- p1183

Oates, B. J. (2006), *Researching Information Systems and Computing*, Sage Publications, London

Over, J. W.(2002), 'Team Software Process for Secure System Development', *Carnegie Mellon, Software Engineering Institute*, Available at: <http://www.sei.cmu.edu/tsp/publications/tsp-secure.pdf> (Last Accessed: August 2011)

OWASP. (2008), 'Threat Risk Modelling', *OWASP*,
Available at: http://www.owasp.org/index.php/Threat_Risk_Modeling

OWASP (2010) 'OWASP Top 10 – 2010 The Ten Most Critical Web Application Security Risk', The OWASP Foundation, Available at: https://www.owasp.org/index.php/Top_10_2010

Paul, M (2011), 'Software Security, Being Secure in an Insecure World' *Software Community (ISC)² Whitepapers*, Available at:
https://www.isc2.org/uploadedFiles/%28ISC%292_Public_Content/Certification_Programs/CSSLP/CSSLP_WhitePaper_3B.pdf (Last Accessed: March 2012)

Pemmaraju, K., Lord, E. and McGraw, G., (2000), 'Software Risk Management: The importance of building quality and reliability into the full development lifecycle', *Cigital, Inc.*, Available at: www.cigital.com/whitepapers/dl/wp-qandr.pdf

Piessens, F. (2002), 'A taxonomy of causes of software vulnerabilities in internet software', Supplementary Proceedings of the 13th International Symposium on Software Reliability Engineering, pp47-52

Pomraning, M. J(2005), 'Injection Flaws: Stop Validating Your Input', *Black Hat 2005*, USA, Available at: <http://www.blackhat.com/presentations/bh-usa-05/bh-us-05-pomraning-update.pdf> (Last Accessed March 2012)

Ralston, P.A.S; Graham, J.H. and Hieb, J. L. (2007), 'Cyber security risk assessment for SCADA and DCS networks', *ISA Transaction*, Vol.46(4), pp583-594

Redwine, S. T. Jr and Davis, N.; et al, (2004), 'Process To Produce Secure Software: Towards more Secure Software', *National Cyber Security Summit*, Vol. 1

Ricard, R. (2011), 'ISO 1799 Risk Analysis Toolkit', Available at:
<http://sourceforge.net/projects/ratiso17799> (Last Accessed: July 2011)

Richardson, R. (2007), 'CSI Survey 2007, the 12th Annual Computer Crime and Security Survey', *Computer Security Institute*, Available at:
http://gocsi.com/sites/default/files/uploads/2007_CSI_Survey_full-color_no%20marks.indd_.pdf (Last Accessed: April 2013)

Ryan, J., Lin, M., and Mikkulainen, R., (1998), 'Intrusion Detection with Neural Networks,' *Advances in Neural Information Processing Systems*, vol. 10, MIT Press, pp943-949

Schumacher, et.al, (2006) 'Security Patterns: Integrating Security and System Engineering' *John Wiley & Sons, Ltd*, Chichester UK

Shulman, A. (2006), 'Top Ten Database Security Threats: How to Mitigate the Most Significant Database Vulnerabilities' Available at: http://www.schell.com/Top_Ten_Database_Threats.pdf (Last Accessed: March 2012)

Smith, L. (2003) An Introduction to Neural Networks, *Centre for Cognitive and Computational Neuroscience*, <http://www.cs.stir.ac.uk/~lss/NNIntro/InvSlides.html> (Last Accessed: May 2011)

Spampinato, D. G. (2008), 'SeaMonster: Providing Tool Support for Security Modelling', NISK Conference, Available at: http://www.shieldsproject.eu/files/docs/seamonster_nisk2008.pdf (Last Accessed: November 2011)

Srinivasa, K.D. and Sattipalli, A. R, (2009) Hand Written Character Recognition using Back Propagation Network, *Journal of Theoretical and Applied Information Technology*, Vol. 5(3), pp 257-269

Steel, C., et.al (2005) 'Core Security Patterns: Best Practices and Strategies for J2EE, Web Services and Identity Management' *Pearson Education, Inc.*, Massachusetts, USA.

Stephens, M., (2003) The Case Against Extreme Programming, Software reality, January 2003, http://www.softwarereality.com/lifecycle/xp/case_against_xp.jsp (Last Accessed: March 2012)

Stergiou, C. and Siganos, D. (1997) Neural Network, *Surprise 96 Journal*, Vol.4

Swigart, S and Campell, S. (2008), '*Threat Modelling at Microsoft*', Available at: http://download.microsoft.com/download/6/9/B/69BCB7C6-D158-4073-AD3E-F849E8ACBCE0/SDL_Series_-_4.pdf (Last Accessed: November 2011)

Telang, R. and Wattal, S.(2004), 'Impact of Software Vulnerability Announcement on Market Value of Software Vendors- an Empirical Investigation', *Workshop on Information Systems and Economics (WISE)*, Available at:http://infoecon.net/workshop/pdf/telang_wattal.pdf (Last Accessed: June 2011)

Tessey, G. (2002), 'The Economic Impacts of Inadequate Infrastructure for software Testing', *National Institute of Standards and Technology (NIST)*, Available at: <http://www.nist.gov/director/prog-ofc/report02-3.pdf> (Last Accessed: August 2011)

Turchin, V. F., (1977) 'The Phenomenon of Science, a Cybernetic Approach to Human Evolution' *Columbia University Press*, New York

Vaishnavi, V. and Kuechler, B. (2009), 'Design Research in Information Systems', *Association for Information Systems* Available at: <http://home.aisnet.org/displaycommon.cfm?an=1&subarticlenbr=279> (Last Accessed: March 2012)

Viega, J. and McGraw, G. (2002), '*Building Secure Software*', *Addison-Wesley*, USA

Wiesauer A. and Sametinger J. A (2009), 'Security Design Pattern Taxonomy Based On Attack Patterns: Findings of a Systematic Literature Review', *Proceedings of the International Conference on Security and Cryptography*

Wiseman, S. (2006), *Software Security and the Software Development Lifecycle*, *Booz Allen Hamilton*, Available at: <http://www.issanova.org/Documents/ArchivePresentations/Presentation-5.2006-Wisseman.ppt> (Last access date: May 2009)

Yu, E. et.al (2007) A Social Ontology for integrating security and software engineer In *Integrating Security Software Engineering: Advances and Visions*, *Idea Group Publishing*, UK, pp70-105

Zachman, J.A. (1987) 'A Framework for Information System Architecture' *IBM System Journal*, Vol.26 (3), pp276-292

Appendix

Appendix I

Security Pattern by Blakely et.al (2004)

Available System Security Pattern

Pattern Name	Description
Checkpoint System pattern	<ul style="list-style-type: none">• Used to recover and restore a system to a known valid state in case a component fails• Offers protection from loss or corruption of state information
Standby Pattern	<ul style="list-style-type: none">• Used to resume service provided by one component from another component• Offers backup when a component fails and cannot be recovered. An similar or identical component is used to provide continues services
Comparator-Check Fault Tolerant System pattern	<ul style="list-style-type: none">• Used to design a system so that can detect an independent failure of one component quickly and not cause a system-wide failure.
Replication System pattern	<ul style="list-style-type: none">• Used to create multiple points of presence and recovery in the case of the failure of one or more components or links.• Provide a means for load balancing and redirection to decrease the chance of non-availability
Error Detection\Correction pattern	<ul style="list-style-type: none">• Used to add redundancy to data to facilitate later detection and recovery of error• Offers protection against data corruption by deducing errors and possibly correcting them in order to ensure correct information exchange or stored.

Protected System Security Pattern

Pattern Name	Description
Protected System pattern	<ul style="list-style-type: none">• Used to provide structure through which all access by clients is mediated by a guard which must be by-passed.• Enforces security policy by controlling access to resources according to predefined policy
Policy Pattern	<ul style="list-style-type: none">• Used to isolate policy enforcement to a discrete component.• Ensures that policy enforcement are performed in the proper sequence• “An authenticated user owns a security context (e.g. a role) that is passed to the guard of resource. The guard checks inside the policy whether the context of this user and the rules match and provides or denies access to the resource”
Authenticator pattern	<ul style="list-style-type: none">• Used to perform authentication of a requesting process before deciding access to distributed object
Subject Descriptor patter	<ul style="list-style-type: none">• Used to provide access to security attributes of an entity on whose behalf operations are to be performed.• Used to control the conditions under which authorization is to be performed.• Used to represent authorization subjects as sets of predicates or assertions on attributes and property values
Secure Communication pattern	<ul style="list-style-type: none">• Used to secure the communication of two parties in the presence of threats.• It us used to ensure that the mutual security objectives are met

Security Context pattern	<ul style="list-style-type: none"> • Provides container for security attributes and data relating to execution context, process, operation or action.
Security Association pattern	<ul style="list-style-type: none"> • Defines a structure which provides each participant in a secure communication with the information it will use to protect messages to be transmitted to the other party. • Also provide participant with information needed to understand and verify the protection applied from the other party.
Secure Proxy pattern	<ul style="list-style-type: none"> • Defines the relationship between the guards of two instances of protected system in the case when one instance is entirely contained within the other. • Can be used to provide defence in depth

Appendix II

Security pattern by Steel et.al (2005)

Web Tier Security Patterns

Pattern Name	Description
Authentication Enforcer	This pattern shows how a browser client should authenticate with the server. It creates a base Action class to handle authentication of HTTP requests.
Authorization Enforcer	This pattern creates a base Action class to handle authorization of HTTP requests.
Intercepting Validator	This pattern refers to secure mechanisms for validating parameters before invoking a transaction. Unchecked parameters may lead to buffer overrun, arbitrary command execution, and SQL injection attacks. The validation of application-specific parameters includes validating business data and characteristics such as data type (string, integer), format, length, range, null-value handling, and verifying for character-set, locale, patterns, context, and legal values.
Secure Base Action	The secure base action is a pattern for centralizing and coordinating security-related tasks within the Presentation Tier. It serves as the primary entry point into the Presentation Tier and should be extended, or used by a Front Controller. It coordinates use of the Authentication Enforcer, Authorization Enforcer, Secure Session Manager, Intercepting Validator, and Secure Logger to ensure cohesive security architecture throughout the Web Tier.
Secure Logger	This pattern defines how to capture the application-specific events and exceptions in a secure and reliable manner to support security auditing. It accommodates the different behavioral nature of HTTP servlets, EJBs, SOAP messages, and other middleware events.
Secure Pipe	This pattern shows how to secure the connection between the client and the server, or between servers when connecting between trading partners. In a complex distributed application environment, there will be a mixture of security requirements and constraints between clients, servers, and any intermediaries. Standardizing the connection between external parties using

Pattern Name	Description
	<p>the same platform and security protection mechanism may not be viable.</p> <p>It adds value by requiring mutual authentication and establishing confidentiality or non-repudiation between trading partners. This is particularly critical for B2B integration using Web services.</p>
Secure Service Proxy	<p>This pattern is intended to secure and control access to J2EE components exposed as Web services endpoints. It acts as a security proxy by providing a common interface to the underlying service provider components (for example, session EJBs, servlets, and so forth) and restricting direct access to the actual Web services provider components. The Secure Service Proxy pattern can be implemented as a Servlet or RPC handler for basic authentication of Web services components that do not use message-level security.</p>
Secure Session Manager	<p>This pattern defines how to create a secure session by capturing session information. Use this in conjunction with Secure Pipe. This pattern describes the actions required to build a secure session between the client and the server, or between the servers. It includes the creation of session information in the HTTP or stateful EJB sessions and how to protect the sensitive business transaction information during the session.</p> <p>The Session pattern is different from the Secure Session Manager pattern in that the former is generic for creating HTTP session information. The latter is much broader in scope and covers EJB sessions as well as server-to-server session information.</p>
Intercepting Web Agent	<p>This pattern helps protect Web applications through a Web Agent that intercepts requests at the Web Server and provides authentication, authorization, encryption, and auditing capabilities.</p>

Business Tier Security Patterns

Pattern Name	Description
Audit Interceptor	<p>The Secure Logger pattern provides instrumentation of the logging aspects in the front, and the Audit Interceptor pattern enables the administration and manages the logging and audit in the back-end.</p>
Container Managed Security	<p>This pattern describes how to declare security-related information for EJBs in a deployment descriptor.</p>
Dynamic Service Management	<p>This pattern provides dynamically adjustable instrumentation of security components for monitoring and active management of business objects.</p>
Obfuscated	<p>This pattern describes ways of protecting business data represented in transfer</p>

Pattern Name	Description
Transfer Object	objects and passed within and between logical tiers.
Policy Delegate	This pattern creates, manages, and administers security management policies governing how EJB tier objects are accessed and routed.
Secure Service Façade	<p>This pattern provides a session façade that can contain and centralize complex interactions between business components under a secure session. It provides dynamic and declarative security to back-end business objects in the service façade. It shields off foreign entities from performing illegal or unauthorized service invocation directly under a secure session.</p> <p>Session information can be also captured and tracked in conjunction with the Secure Logger pattern.</p>
Secure Session Object	This pattern defines ways to secure session information in EJBs facilitating distributed access and seamless propagation of security context.

Web Services Tier Security Patterns

Pattern Name	Description
Message Inspector	This pattern checks for and verifies the quality of XML message-level security mechanisms, such as XML Signature and XML Encryption in conjunction with a security token. The Message Inspector pattern also helps in verifying and validating applied security mechanisms in a SOAP message when processed by multiple intermediaries (actors). It supports a variety of signature formats and encryption technologies used by these intermediaries.
Message Interceptor Gateway	This pattern provides a single entry point and allows centralization of security enforcement for incoming and outgoing messages. The security tasks include creating, modifying, and administering security policies for sending and receiving SOAP messages. It helps to apply transport-level and message-level security mechanisms required for securely communicating with a Web services endpoint.
Secure Message Router	This pattern facilitates secure XML communication with multiple partner endpoints that adopt message-level security and identity-federation mechanisms. It acts as a security intermediary component that applies message-level security mechanisms to deliver messages to multiple recipients where the intended recipient would be able to access only the required portion of the message and remaining message fragments are made confidential.

Security Patterns for Identity Management and Service Provisioning

Pattern Name	Description
Assertion Builder	This pattern defines how an identity assertion (for example, authentication assertion or authorization assertion) can be built.
Credential Tokenizer	This pattern describes how a principal's security token can be encapsulated, embedded in a SOAP message, routed, and processed.
Single Sign-on (SSO) Delegator	This pattern describes how to construct a delegator agent for handling a legacy system for single sign-on (SSO).
Password Synchronizer	This pattern describes how to securely synchronize principals across multiple applications using service provisioning.

Appendix III

Security Design Patterns by Kinezle and Elder (2003)

Structural Patterns

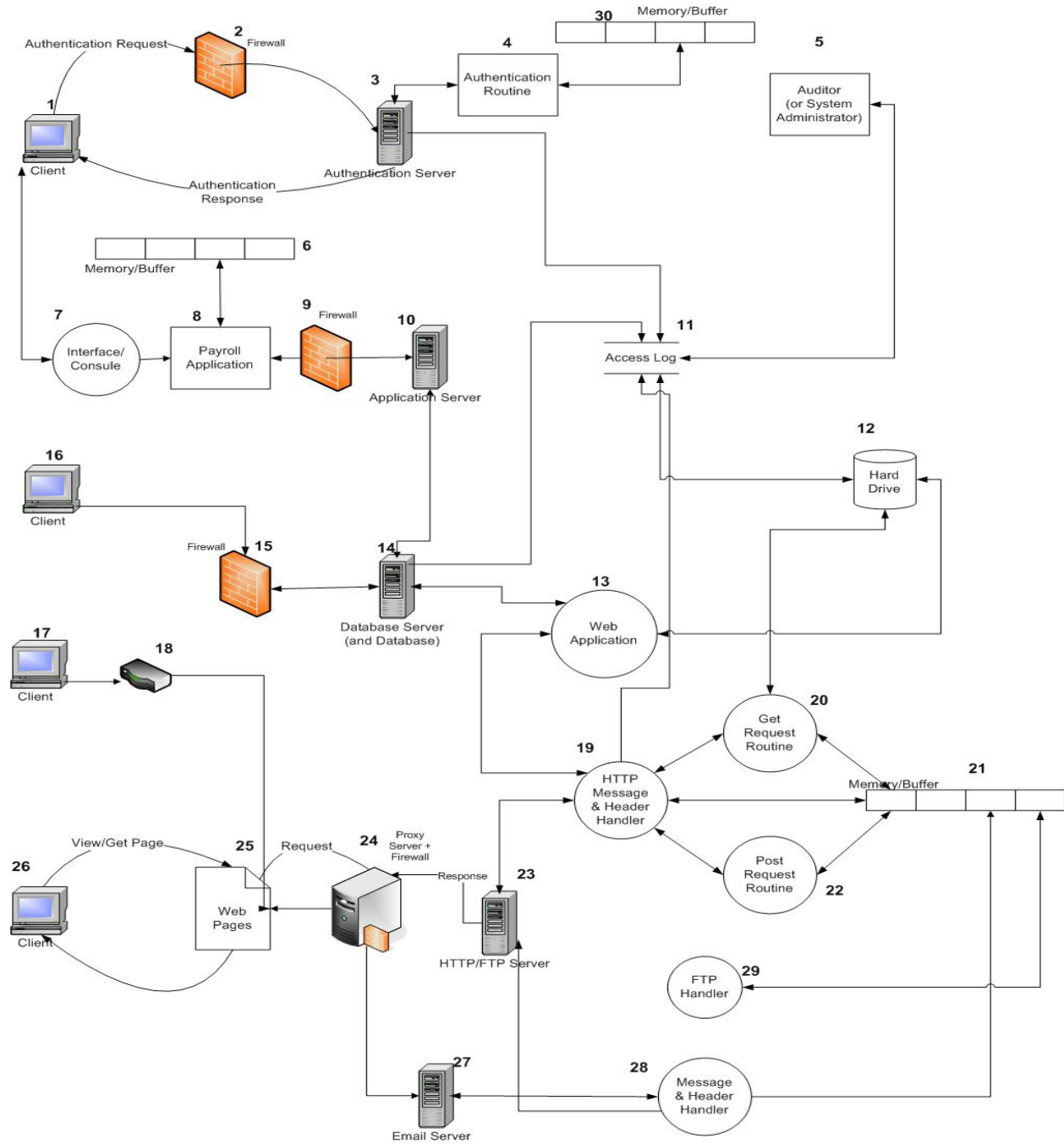
Pattern Name	Description
Account Lockout	Passwords are the only approach to remote user authentication that has gained widespread user acceptance. However, password guessing attacks have proven to be very successful at discovering poorly chosen, weak passwords. Worse, the Web environment lends itself to high-speed, anonymous guessing attacks. Account lockout protects customer accounts from automated password security guessing attacks, by implementing a limit on incorrect password attempts before further attempts are disallowed.
Authenticated Session	An authenticated session allows a Web user to access multiple access-restricted pages on a Web site without having to re-authenticate on every page request. Most Web application development environments provide basic session mechanisms. This pattern incorporates user authentication into the basic session model.
Client Data Storage	It is often desirable or even necessary for a Web application to rely on data stored on the client, using mechanisms such as cookies, hidden fields, or URL parameters. In all cases, the client cannot be trusted not to tamper with this data. The <i>Client Data Storage</i> pattern uses encryption to allow sensitive or otherwise security-critical data to be securely stored on the client.
Client Input Filters	Client input filters protect the application from data tampering performed on untrusted clients. Developers tend to assume that the components executing on the client system will behave as they were originally programmed. This pattern protects against subverted clients that might cause the application to behave in an unexpected and insecure fashion.
Directed Session	The <i>Directed Session</i> pattern ensures that users will not be able to skip around within a series of Web pages. The system will not expose

	multiple URLs but instead will maintain the current page on the server. By guaranteeing the order in which pages are visited, the developer can have confidence that users will not undermine or circumvent security checkpoints.
Hidden Implementation	The <i>Hidden Implementation</i> pattern limits an attacker's ability to discern the internal workings of an application—information that might later be used to compromise the application. It does not replace other defenses, but it supplements them by making an attacker's job more difficult.
Encrypted Storage	The <i>Encrypted Storage</i> pattern provides a second line of defense against the theft of data on system servers. Although server data is typically protected by a firewall and other server defenses, there are numerous publicized examples of hackers stealing databases containing sensitive user information. The <i>Encrypted Storage</i> pattern ensures that even if it is stolen, the most sensitive data will remain safe from prying eyes.
Minefield	The <i>Minefield</i> pattern will trick, detect, and block attackers during a break-in attempt. Attackers often know more than the developers about the security aspects of standard components. This pattern aggressively introduces variations that will counter this advantage and aid in detection of an attacker.
Network Address blacklist	A network address blacklist is used to keep track of network addresses (IP addresses) that are the sources of hacking attempts and other mischief. Any requests originating from an address on the blacklist are simply ignored. Ideally, breaking attempts should be investigated and prosecuted, but there are simply too many such events to address them all. The <i>Network Address Blacklist</i> pattern represents a pragmatic alternative.
Partitioned Application	The <i>Partitioned Application</i> pattern splits a large, complex application into two or more simpler components. Any dangerous privilege is restricted to a single, small component. Each component has tractable security concerns that are more easily verified than in a monolithic application.
Password Authentication	Passwords are the only approach to remote user authentication that has gained widespread user acceptance. Any site that needs to reliably identify its users will almost certainly use passwords. The <i>Password Authentication</i> pattern protects against weak passwords, automated password-guessing attacks, and mishandling of passwords.
Password Propagation	Many Web applications rely on a single database account to store and manage all user data. If such an application is compromised, the attacker might have complete access to every user's data. The <i>Password Propagation</i> pattern provides an alternative by requiring that an individual user's authentication credentials be verified by the database before access is provided to that user's data.
Secure Assertion	The <i>Secure Assertion</i> pattern sprinkles application-specific sanity checks throughout the system. These take the form of <i>assertions</i> – a popular technique for checking programmer assumptions about the environment and proper program behavior. A secure assert maps conventional

	assertions to a system-wide intrusion detection system (IDS). This allows the IDS to detect and correlate application-level problems that often reveal attempts to misuse the system.
Server Sandbox	Many site defacements and major security breaches occur when a new vulnerability is discovered in the Web server software. Yet most Web servers run with far greater privileges than are necessary. The <i>Server Sandbox</i> pattern builds a wall around the Web server in order to contain the damage that could result from an undiscovered bug in the server software.
Trusted Proxy	A trusted proxy acts on behalf of the user to perform specific actions requiring more privileges than the user possesses. It provides a safe interface by constraining access to the protected resources, limiting the operations that can be performed, or limiting the user's view to a subset of the data.
Validated Transaction	The <i>Validated Transaction</i> pattern puts all of the security-relevant validation for a specific transaction into one page request. A developer can create any number of supporting pages without having to worry about attackers using them to circumvent security. And users can navigate freely among the pages, filling in different sections in whatever order they choose. The transaction itself will ensure the integrity of all information submitted.

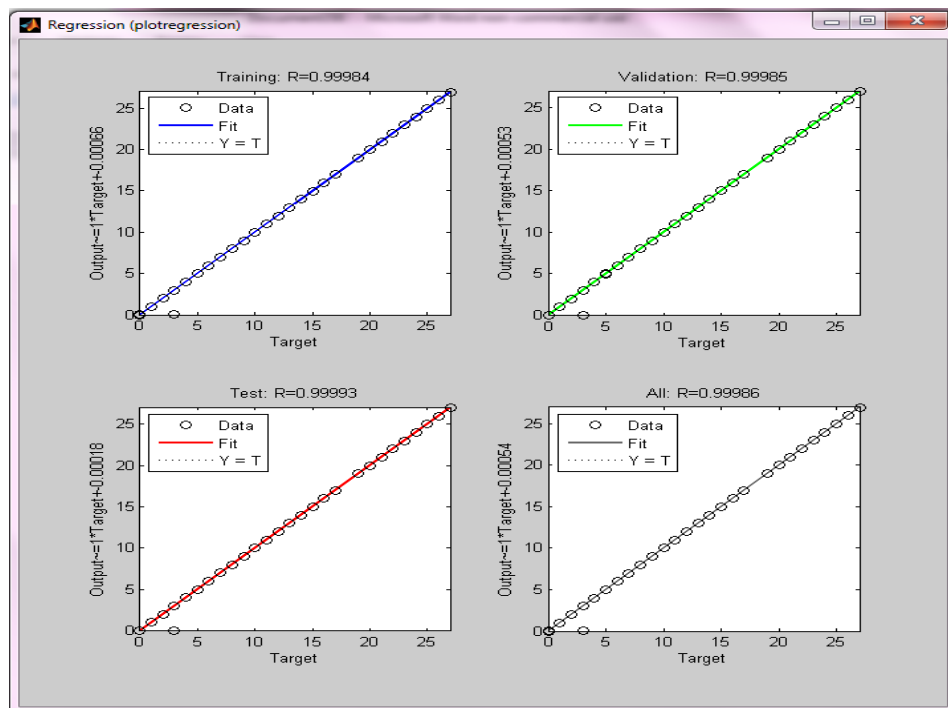
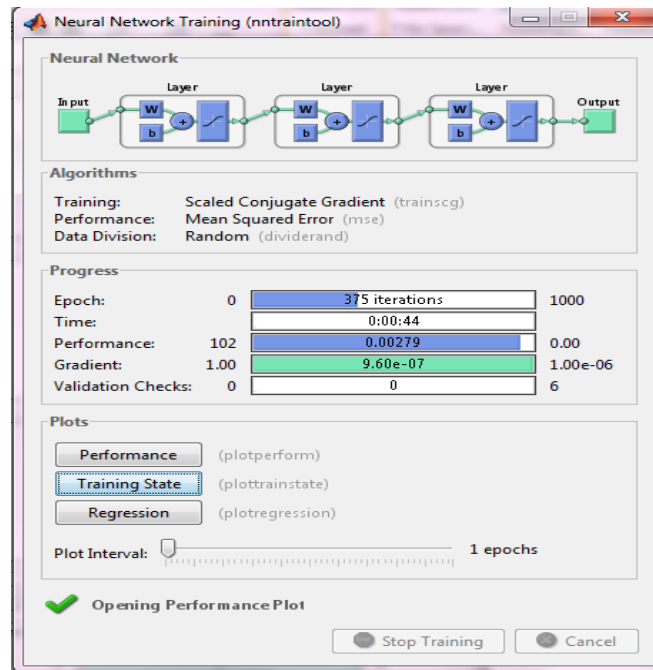
Appendix IV

System Design of a hypothetical banking system

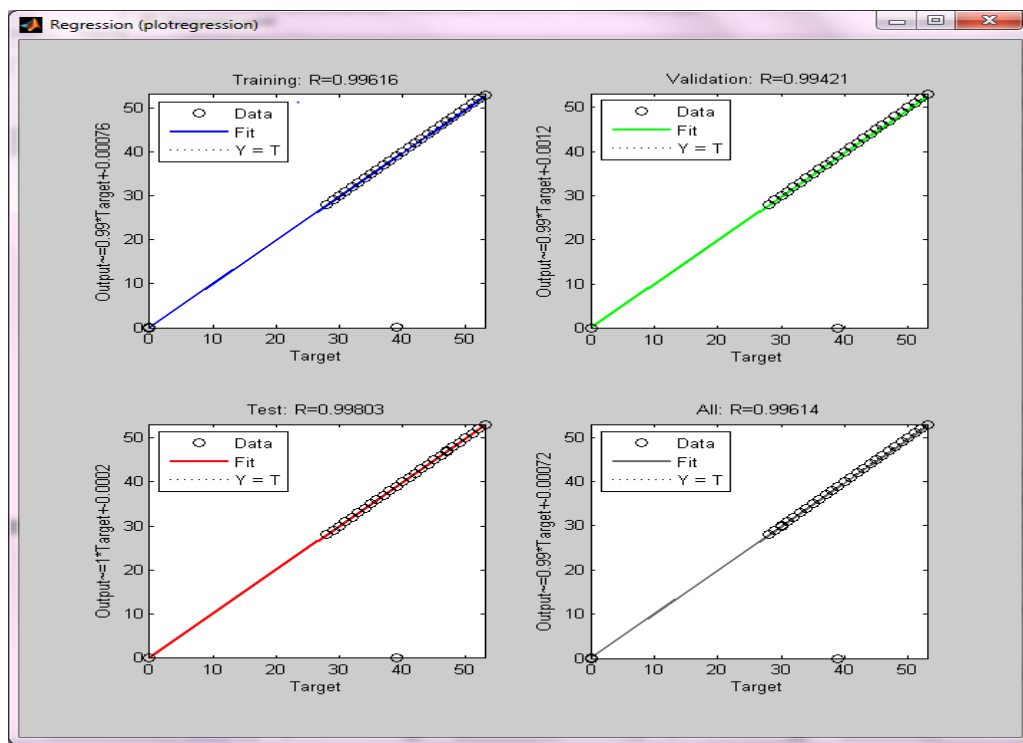
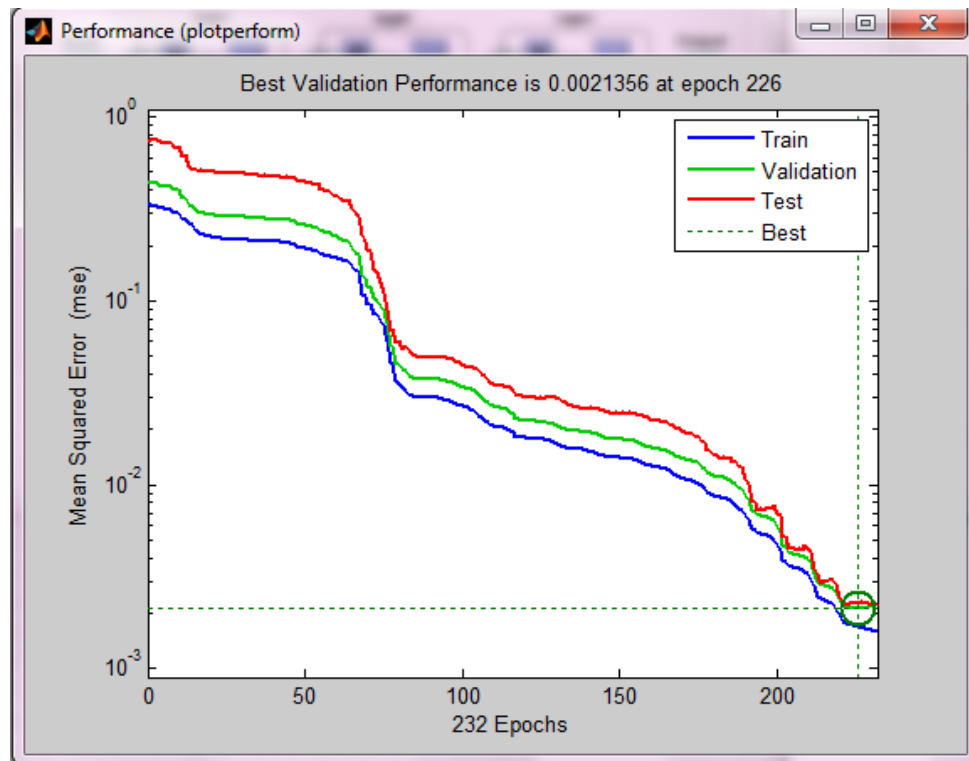


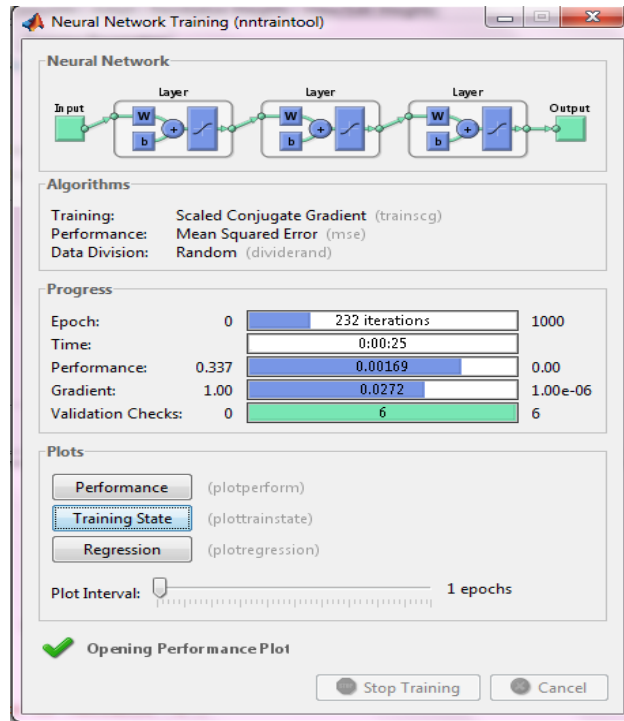
Appendix V

Screenshots of Neural Network I (first Network)

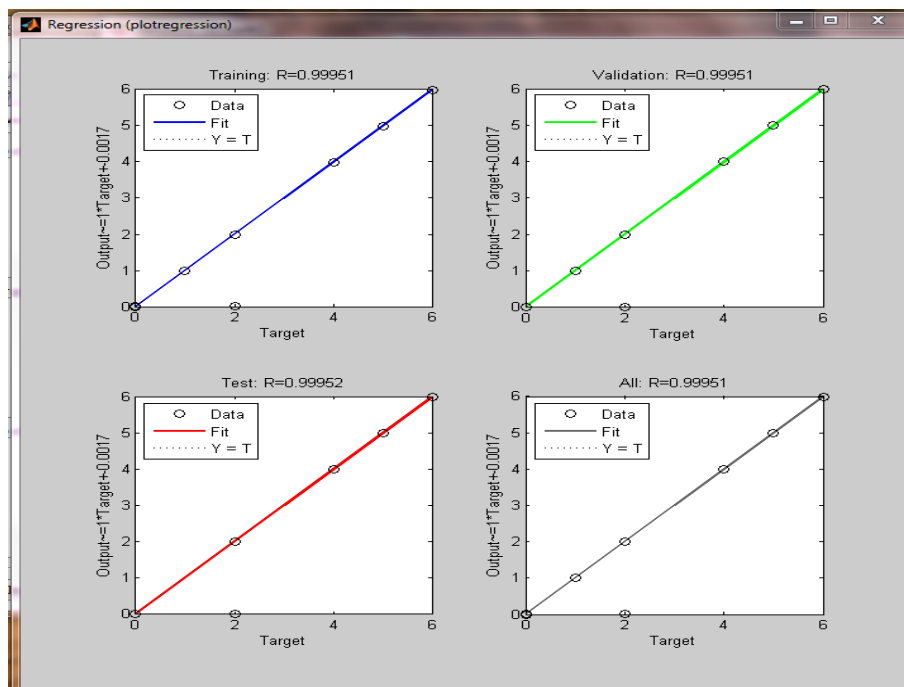


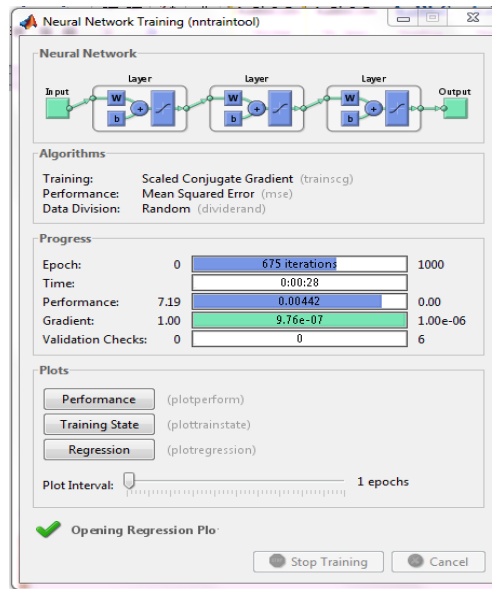
Screenshots of Neural Network I (Second Network)





Screen Shot of Neural Network II





Appendix VI

Neural Network 1 (First Network: Weight to layer 3)

[-0.16874 0.46912 -0.17275 0.93263 0.22674 0.33647 0.82078 -1.2175 -0.30718 0.1619 0.60244 -
0.84596 0.35917 -0.58197 -0.1355 0.78296 -0.14891 -0.63678 0.58449 0.60938 0.74851 -0.75645 -
0.50426 -0.86166 -0.27702 0.10394 0.49177 -0.1605 1.0453 -0.012048 0.19877 0.67603 -0.11851 -
0.043609 -0.89364;

-0.37075 0.60491 0.36731 2.1922 -0.6221 0.12072 1.0724 0.35546 -2.0527 -0.53752 -1.4685 -0.022488
0.64421 0.83976 -0.50733 0.27778 0.17341 -1.0992 0.81255 0.39671 0.70053 -0.081449 -0.59588 -
0.28674 -0.24363 0.46533 0.21334 -0.21313 1.7439 0.49439 -0.040434 -0.42118 -0.82257 0.98078 -
0.45328;

-0.1083 0.40013 0.75861 0.43049 -1.2952 -0.56643 -0.061328 -0.54782 1.2521 0.12715 -2.1861
0.47863 -0.053031 -0.78504 -0.43256 0.43859 1.068 -0.46446 -2.2287 0.88026 0.65504 0.95359 -
0.84473 -1.0891 -0.68932 0.65899 0.2167 -1.5766 -0.18785 0.34162 -0.91247 -0.042785 -0.35586 -
0.58015 -0.47937;

-0.45063 0.50104 -0.28596 0.40768 -0.91112 1.7644 0.10891 -0.26777 -1.981 1.2315 0.13444 0.1784
0.69268 -1.3179 -0.48735 1.0694 2.0058 -0.55395 -0.55371 0.91008 1.0734 0.10779 -0.58433 -0.78402
0.23368 -0.41306 0.8539 -0.53076 1.2675 -2.2481 -0.63883 0.33174 0.46121 0.5084 0.9367;

0.91631 0.07302 -1.2417 -0.41323 1.3863 -0.023949 -1.3495 0.89421 0.95998 -0.11751 -0.99684
1.3973 0.59484 1.5423 -0.28783 0.86094 -0.089169 -0.85489 2.4052 0.56013 0.52985 1.7752 -0.4175 -
0.35226 1.2106 0.23597 0.37343 0.98609 -0.23339 0.43199 -0.39431 0.19539 -1.2496 1.6346 -0.50527;

0.60016 0.29291 -0.44971 0.78497 -0.48747 -0.29152 -0.30941 0.48076 -0.10347 -0.0019005 0.14059
0.53035 0.58725 -0.074515 -0.52416 0.045644 -0.01699 -0.4462 1.1127 0.024181 -0.06964 -0.51163
0.10999 0.012175 0.37395 1.3187 0.0046624 -0.67747 -0.94985 -0.44251 -1.0815 -0.057936 0.21611 -
0.037913 -0.673;

-0.93018 0.3447 -0.30622 0.8012 -0.84328 -0.63199 0.29164 -0.63001 -0.3383 0.31774 0.012506 -1.536
0.89533 -0.12367 -1.0244 0.54619 0.57499 -0.2583 -0.99891 0.28656 0.65137 1.0307 -0.69852 -0.27886
1.0703 -0.431 0.19843 -0.68107 -0.78461 -1.2297 -0.28078 -0.021021 0.30103 0.035136 -3.2589;

2.3712 -0.020569 1.0805 0.061385 0.47081 2.9245 0.12073 0.116 -1.2281 0.11631 -0.015111 0.18819
0.075333 1.3236 -0.42198 0.1665 0.13483 -0.80836 -0.40835 0.62561 0.56523 -1.016 0.088189 -
0.098191 1.3194 1.0351 0.040431 -0.20273 0.55845 -0.84564 0.036098 0.54485 -0.17828 0.39558
0.011753;

-0.83361 0.037664 -1.7883 0.97998 1.6225 -1.9614 -0.60463 -0.79921 -0.44931 -1.5775 -0.064096 -
1.2352 0.37646 -0.89299 -0.5847 1.0306 -0.43719 -1.1248 -0.33434 0.7354 0.88145 0.80174 0.41088 -
0.96219 0.43431 0.47466 0.45816 -0.40462 1.2164 -1.7696 -0.98476 0.95398 0.19492 0.66361 0.18371;

1.9634 -0.3596 -0.48624 0.52468 -1.5157 -1.5481 -0.75522 -0.60929 0.21619 -0.84161 -0.44918 -1.3449
0.58941 1.9428 -0.57106 0.9087 -0.59144 -0.25726 -0.11953 0.27224 0.35514 1.3963 -0.19481 -0.75189
-0.66517 -0.65553 0.45938 -0.81879 -0.57675 -1.1529 -0.71389 0.39541 0.6605 0.38264 -0.20745;

1.3363 -0.095471 -0.44977 0.68648 1.1049 -0.18412 -0.50327 0.35878 -0.51511 -0.66366 -1.3242
0.56085 0.33476 -0.83207 -0.41446 0.7586 -0.28156 -0.25522 -0.8105 0.41925 0.2915 0.25792 -0.279 -
0.38436 0.40188 0.61716 -0.075181 -0.71581 0.22185 0.71126 -0.46072 0.46842 1.5155 0.31407 -
0.25532;

0.005081 -0.020256 -0.12506 -0.137 -0.60889 -0.066088 0.781 -0.20534 0.77267 -0.47274 0.37117 -
0.20639 -0.12378 -0.29893 0.013001 0.3097 0.010454 -0.16373 -1.0239 0.51806 0.35029 -0.093545
0.15633 -0.62207 1.0679 0.53538 1.2272 -0.43737 -0.40301 -0.15853 0.081084 1.5515 0.85398
0.067091 -0.30354;

-0.27075 0.5451 -2.0196 -1.3172 -1.1419 -1.517 -0.93752 -1.2761 -1.8635 0.95251 1.5891 0.0037349
0.39957 1.1789 -0.54893 1.0023 0.48079 -0.32365 -0.90432 0.98014 0.99491 -0.19345 -0.82439 -0.1963
0.09139 0.17318 0.423 -0.63343 0.53891 2.6909 -0.3045 0.15552 -0.49413 0.12658 0.0019409;

0.57914 2.1806 -0.11365 0.19539 0.27843 -0.55773 -0.0019489 -0.5663 -0.10024 -0.42955 -0.077741 -
0.36 0.14317 -0.28479 -0.51904 0.32154 0.084153 0.082452 -0.63545 -0.050958 0.19833 -0.72762 -
1.5764 -0.52302 0.46164 0.39028 -0.06375 -0.61024 -0.38948 -0.36031 0.15571 -0.16362 -0.21822 -
0.052815 -0.067036;

0.26184 0.64629 0.20241 -1.8688 0.55728 0.33182 -0.5613 -1.499 0.53433 0.329 -1.0948 0.099771
0.94483 -1.2848 -0.57714 0.6661 -0.10247 -0.20858 0.58051 0.40007 0.41128 0.10002 -0.44536 -0.8396
-0.15608 -0.043278 1.2195 0.048752 -0.33135 0.067516 -0.1272 -0.3257 0.87144 -0.17051 -0.14156;

1.0996 0.37021 0.45179 0.39903 -0.51302 0.13948 1.4701 -0.9589 0.43698 0.62273 -0.99534 0.13125
0.52355 1.0193 -0.36298 0.86183 -0.98225 -0.52502 0.20888 0.25955 0.3902 -0.53294 -0.26033 -
0.90482 1.8221 0.1055 0.01444 -0.62337 0.021385 -0.27247 0.21679 -0.44207 0.88962 0.70243 -
0.036662;

-0.29616 0.67145 0.18642 0.22566 1.129 -0.020646 0.69462 -0.024119 -0.29747 0.73282 0.11451 -
0.55373 0.68005 0.077157 -0.16139 0.28083 -0.83718 -0.27889 -0.64558 0.72196 0.059226 0.41305 -
0.42 -0.4984 1.271 0.28179 0.47993 -2.0659 0.22861 0.72645 -0.28891 -0.024572 -0.57331 0.10878 -
0.14352;

-1.0713 0.47274 -0.16907 -2.6718 0.79298 1.8399 -0.25943 0.082362 0.0053939 -1.0884 -0.24043
0.7413 0.30837 -0.63532 -0.92624 0.30116 -0.37611 -0.84821 -1.078 0.71424 0.12971 0.085605 -
0.26007 -0.9068 0.43572 -0.22903 0.33892 -0.73762 0.24666 -1.6835 -0.67711 -0.35032 -0.47052 -
0.35499 -0.41858;

-0.39696 0.39997 0.26203 0.50923 1.21 -0.28958 0.39347 0.014213 2.6689 -1.8635 1.2526 0.71566
0.48233 1.8037 -0.34577 0.66118 1.4289 -0.7266 -2.0338 0.91744 1.0442 -0.080937 -0.79409 -0.97551
0.094203 0.91673 0.20046 -0.21624 -0.4189 1.3897 -0.40207 0.19589 -0.68227 0.12572 -0.72807;

-0.41681 -0.41505 -1.5604 -1.2439 0.93151 -0.2496 1.4941 0.16361 0.28889 0.12021 0.1022 -0.86283
0.30597 1.2326 -0.16369 0.091447 -0.48314 -0.40747 0.86488 0.41102 0.27273 -1.0703 -0.92711 -
0.80791 0.31456 0.61888 0.18815 -0.47133 -1.3027 -1.0286 0.21561 -0.25669 -0.23219 -0.14272 -
0.18964;

-0.34271 0.32036 -2.1063 1.0928 1.0386 3.1617 0.080611 -0.23116 1.8746 0.82869 0.29223 1.5338
0.42698 -0.64105 -0.2654 0.69486 -0.94791 -0.55473 -0.49423 0.10671 0.79908 0.28972 -0.46165 -
0.30679 0.13562 -0.091318 0.21584 -0.13459 -0.35401 1.2973 -0.41617 0.75114 0.063945 0.33344 -
0.77448;

-0.16133 0.3878 0.062173 -1.1818 0.41922 0.075442 0.93286 -0.19557 -0.67739 0.071846 -0.57973
0.49595 0.40671 0.23613 -0.90741 0.33997 -0.32956 -0.82821 0.40086 -0.044961 0.11563 -0.0084878 -
0.3041 -0.16512 -0.11949 0.40711 -0.45734 -0.82926 0.32667 0.11749 -1.6161 1.2322 1.0187 0.61073 -
0.92963;

-0.089573 0.20997 0.49355 -0.83159 1.1258 1.5587 -0.0066089 -0.026393 1.3047 0.31986 2.01 -2.148
0.22348 -0.14294 -0.71449 0.74849 1.6712 -0.61445 2.545 0.39031 0.6314 -0.083588 -0.61346 -0.14176
0.37991 -0.18981 0.10641 -0.83907 0.30997 1.0725 -0.16124 0.31785 0.058244 -0.28293 -0.5086;

1.0678 0.2165 0.47577 -1.6904 -0.13782 0.15952 1.9122 0.13399 0.47221 -0.91265 -1.0237 -0.67731
0.67053 0.026892 -0.21008 0.80735 0.41187 -0.12929 0.35055 0.3811 0.79882 1.283 -0.59784 -0.64256
0.30512 -0.27308 0.46733 -0.69773 -2.236 1.1542 -0.46544 0.023994 -0.33677 1.3884 0.057178;

-0.80474 0.49505 0.40314 -1.4483 -0.77715 0.11736 -1.2416 -0.28978 0.24278 0.4233 0.05443 -0.5929
0.48019 0.32133 -0.95371 0.37852 -0.19461 -0.75764 -1.1751 0.59947 0.040578 0.7144 -0.2238 -

0.67192 -0.12821 1.8456 0.025829 -0.87076 -1.2135 -0.81434 -0.51651 0.74512 0.48069 1.1549 -
0.097741;

-0.87395 0.02663 -1.9877 1.168 1.1627 1.9248 -0.49736 -2.0415 -1.379 -1.5317 0.60118 -0.96778
0.21748 1.8683 -0.65136 0.40536 2.6187 -0.30935 1.6731 0.28783 0.56594 -1.2267 -0.16659 -0.71195 -
0.36162 0.26234 0.2141 0.24062 -0.10049 0.4772 -0.20505 0.4144 2.646 0.4699 -0.61364]

Neural Network I (Second Network: weight to layer 3)

[0.00027829 -0.61828 0.73982 -0.40325 -0.28699 0.089942 -0.26725 -0.048234 -0.29411 -0.57788
0.37841 -0.57048 0.59433 0.0094232 1.4085 0.82056 0.80302 -0.1766 -0.54572 0.65269 0.91527 -
0.44104 0.44588 -0.033723 0.27357 0.63858 -0.093854 -0.070888 -0.55713 -0.328 0.25231 0.55114 -
0.95603 -0.62083 -0.29803;

0.069377 -0.54301 -1.0676 -0.33555 -0.37548 0.020985 0.12136 0.24337 0.080117 -0.24181 0.31416 -
0.85271 0.88078 -0.92719 -1.2092 -0.15579 0.20331 -1.1349 0.5568 0.00089946 0.26062 -0.10447
0.66929 -0.86826 0.13782 0.089057 -0.55251 0.031708 0.22604 -0.36976 0.16308 0.40124 -1.2889 -
0.46932 -0.67402;

-0.772 -0.74579 -0.55708 -0.11964 -0.41502 -1.2333 -0.10064 -0.59397 -0.30849 -0.09555 0.42258 -
0.76372 0.48766 0.32933 0.11672 0.29361 -1.3662 0.82167 0.50149 0.51635 0.14896 0.42495 0.85338 -
0.54342 0.30342 0.20106 -1.1115 -0.29523 -0.37487 1.1887 -0.42638 0.3735 0.13716 -0.28454 -
0.90382;

-0.042158 -0.70406 -0.27203 0.12494 -0.59092 1.0419 -0.49495 -0.085801 -0.36588 -0.056737 0.6553 -
0.2243 0.96982 -0.51594 -1.6151 0.15789 -0.21724 0.9266 0.13196 0.29904 0.16845 0.39745 -0.049178
-0.45508 -0.00062244 -1.6337 0.12764 0.26033 0.5166 -0.10963 -0.44981 0.20918 -1.0314 -0.31119 -
0.42226;

-0.29248 -0.79441 0.20935 -0.22569 -0.62937 0.41342 -0.042484 -0.26192 -1.2337 -0.19639 0.13909
0.03175 -0.68431 -0.79355 0.41636 0.32003 0.01192 0.7439 0.45249 0.11059 0.15618 0.17267 -0.74136
0.020536 0.22543 -0.45403 0.21755 -0.076817 -0.61225 -0.29869 -0.057344 0.18794 0.6706 0.060744 -
0.12071;

0.23764 -0.58026 1.946 0.14825 -0.49704 -0.22677 -0.58459 -0.60264 -0.5686 0.057248 0.037745 -
0.37985 0.45205 0.28124 -1.1667 0.93059 0.98203 -0.076015 -0.4089 0.80834 0.52765 -0.083223 -
0.10655 -0.086983 0.45008 0.15332 0.3953 -0.15956 -0.53378 0.40659 -1.2172 0.20758 0.27337 -
0.77231 -0.16307;

-0.04335 -0.33719 -0.2474 0.07895 -0.57564 0.60017 -0.29364 -0.48244 -0.20828 -0.17485 0.15358
0.84556 0.53709 1.0001 -0.57952 0.39579 -0.52672 -0.33391 0.11712 0.65041 -0.53551 -0.37578 -
0.89508 -0.54239 0.3291 0.58018 0.18767 0.015882 -0.55264 1.1173 0.091167 0.56552 -0.46362 -
0.20206 -0.34357;

-0.077813 -0.58706 -0.72498 -0.41533 -0.026086 -1.3767 -0.2494 -0.21149 -0.27717 -0.75952 0.62801
0.49059 0.35203 -1.2625 -0.39778 0.30806 -1.595 1.1816 0.14263 0.49358 -0.25288 0.043795 -0.78796
-0.39474 0.66069 0.5375 0.2377 0.27158 -1.0209 -0.76241 0.40351 0.011313 0.0095843 -0.55385 -
0.31926;

-1.3688 -0.52692 -0.21074 0.40208 -0.49653 -0.23046 -0.036601 -0.034666 -0.39005 -0.58429 0.55206 -
0.3455 -0.29986 0.86999 -0.45262 -1.4591 0.17064 -0.99505 -0.04477 0.2822 0.5848 0.32942 0.36773
0.14258 0.63582 0.27326 0.46517 0.63749 0.079932 -0.12417 -0.5309 -0.63212 -0.57209 -0.63712 -
0.67117;

0.067794 -0.45955 1.1004 -0.19537 -0.46618 0.76897 -0.29954 -0.81603 0.1069 -0.18783 0.6097
0.99482 -1.3654 0.51394 0.18744 0.5299 -1.2854 0.27815 0.15498 0.49157 0.44353 0.30748 0.56847 -
0.7709 -0.09634 -0.90435 -1.657 0.1487 -0.41868 0.068575 -0.37555 0.43382 0.47103 -0.55086 -0.1408;

-0.67012 -0.22183 0.17053 -0.66708 -0.20122 0.32058 0.26367 -0.35742 -0.37191 -0.07881 0.51379 -
0.34766 -0.37487 0.11681 -0.85251 -0.18141 0.47924 0.51668 -0.18332 0.29349 -0.42823 0.38375 -
0.42223 -0.21017 0.20165 -0.22613 -0.20027 -0.76302 0.27343 0.16861 0.30941 0.66949 0.26353 -
0.35203 -0.83811;

-0.30079 -0.45244 -0.36959 -0.38978 -0.23582 -0.02067 0.28523 -0.84013 0.28226 -0.60085 0.53508
1.1112 -1.1861 0.34301 -0.063788 0.63513 -0.15234 -0.93268 0.46464 0.10315 -1.1959 0.94362 0.83562
-0.5205 0.4551 1.1142 -0.98524 -0.078618 -0.40158 -0.61546 -0.26731 0.099991 -0.35746 -0.56976 -
0.17526;

-0.66101 -0.018699 -0.56883 -0.0012508 -0.55096 -0.33889 0.14019 -0.36607 0.17236 -0.29313
0.025477 0.64751 -0.58474 -0.34733 0.52438 0.46734 0.30871 0.72455 0.62809 0.77604 -0.56914
0.46147 0.034382 -0.73263 0.31404 -0.91629 -0.055692 -1.3777 0.019151 0.86271 -0.58024 0.35584
0.38625 -0.76758 -0.078903;

0.51843 -0.2787 0.62731 -0.27979 -0.74681 -0.79144 0.23654 0.23535 -0.28814 -0.3767 0.55501 -
0.16908 -0.531 0.21491 0.21846 1.0959 -1.0659 0.47586 -0.48653 0.66045 0.058875 -0.27149 -0.56728
-0.62132 0.38313 0.68826 0.14781 -0.34041 -0.14872 0.2342 -0.71345 0.34813 -1.1532 -0.78146
0.45557;

-0.14556 0.035575 0.16672 0.75258 -0.45042 0.11183 0.65317 0.012577 -0.39317 -0.12007 0.49021
0.29203 0.68971 0.21316 -0.68572 0.18562 -0.049189 -0.35021 -0.56222 0.36438 0.3429 0.62996
1.3277 -0.70936 -0.10541 0.26723 0.061138 0.071135 -0.51577 -0.012361 0.1055 1.2352 0.39954 -
0.39893 0.15798;

-0.64604 -0.47498 0.72673 -0.03053 -0.41029 0.22052 -0.35207 0.31955 -1.1631 -0.55511 0.48285
1.1915 0.20717 -0.76187 0.14621 0.028162 0.68518 -0.74801 0.45711 -0.017228 -0.22761 -0.18376
0.65665 0.085489 0.50181 -1.0884 0.036299 0.27281 -0.06702 0.31095 -0.20077 0.063445 -0.46803 -
0.70287 0.11671;

-0.63072 -0.72747 0.82577 -0.17257 -0.6292 -0.76144 1.2819 0.34797 0.21417 -0.25439 0.068862
0.048425 0.76377 -0.38199 -0.15612 -0.4449 0.082724 1.2789 0.96198 0.20213 -0.28874 -0.20415
0.4288 -0.37692 0.80417 -0.46659 -0.17325 -0.71745 -1.2611 -0.1396 -0.17267 0.047271 -0.8112 -
0.23361 0.66804;

-0.032531 -0.51265 -0.84962 -0.022925 -0.71711 -0.3469 0.3259 0.49526 -0.6043 -0.43689 0.39368
0.52137 -0.65181 0.0016131 -0.33833 -0.69996 -0.090013 0.84502 -0.04866 0.55146 0.27733 -0.22216 -
0.54231 -0.74236 0.4064 -0.76426 0.21947 0.26436 -0.0082227 -0.25626 -1.4444 0.91716 0.22058 -
0.20923 0.16794;

0.27077 -0.13475 0.36233 0.00070036 -0.63846 -0.59954 -0.31189 -0.72961 -0.53541 -0.63251 0.81766
0.78618 0.64993 0.36273 0.58305 -0.60392 0.11345 -0.78706 -0.82571 0.87904 -0.70411 -0.31088 -
0.44788 -0.30857 0.49811 0.29983 -0.42049 -0.83441 -0.61932 0.4457 -0.35668 0.37259 0.0095759 -
0.36843 0.075976;

-0.046177 -0.65591 -0.78917 -1.282 -0.22137 0.47174 -0.70709 -0.1985 -0.77975 -1.4153 0.87461 -
1.014 0.96835 0.41824 0.51466 -0.54134 -0.6506 0.28421 -0.69889 0.1293 -1.0086 0.24437 1.0033 -
0.78658 0.64594 -0.7705 0.075496 -0.20028 0.2632 -0.21563 0.015409 -0.31215 0.23701 -0.92992 -
0.2783;

-0.66066 -0.40275 -0.871 -0.39093 -0.61749 -0.30408 0.077377 0.84304 0.024565 -0.54618 0.62791
0.13264 -0.36256 -1.129 -0.53815 1.0532 0.80494 -0.73136 0.43335 0.12625 -0.32452 0.49003 -0.86872
-0.59636 -0.099799 0.50785 -0.6051 0.6242 0.16915 0.79516 -1.214 0.27295 0.83352 -0.44227
0.076523;

-0.18227 -0.60278 0.031406 -0.083375 -0.57549 0.36288 0.82774 -0.31386 0.026657 -0.11086 0.32521
-0.25676 0.55848 -1.0784 0.58173 0.61674 -0.20252 -0.37306 -0.67015 0.57362 -0.13422 0.15712 -
0.57902 -0.067274 0.43864 0.33628 -0.89326 0.12463 0.19942 0.52269 0.27112 0.080844 -0.097539
0.16523 -0.46618;

-0.010304 -0.43788 -0.19397 0.27319 -0.42338 0.45441 0.33639 0.18383 -0.52026 -0.1667 0.34489
0.83955 -0.094048 0.070629 0.37573 -0.22091 -0.87595 0.41503 -0.71809 0.64468 -0.020614 0.89486 -
0.1702 -0.54084 0.52514 -0.38192 -0.24907 0.1759 -0.033523 0.64493 0.25924 0.10936 -0.83066 -
0.2541 -1.8277;

-0.4687 -0.84897 -1.2442 0.038577 -0.1589 -2.8048 -1.7591 1.1397 -0.60626 -0.16105 0.90036 -0.27709
0.93857 -1.9632 0.34945 -0.81848 -0.18411 -1.5029 -0.31678 0.25668 0.70859 1.6561 -0.072868 -
0.81981 0.64868 0.48771 0.054691 -2.2841 -1.6095 -0.34523 -0.15382 0.30716 0.024501 -0.38105 -
0.80389;

-0.83169 -0.38409 0.9496 -0.66819 -1.1389 2.8259 2.7076 0.3195 -0.92501 -0.87612 0.28017 -0.42328
0.60949 1.5791 0.40449 0.38076 1.2243 1.5092 -0.18739 0.91793 2.2285 0.57156 -1.4124 -0.53684
0.89181 -1.4472 -0.036904 0.33293 -1.9015 0.26894 0.31814 -0.12805 0.37712 -0.80256 0.18135;

-0.23672 -0.56884 -0.21291 0.13882 -0.46784 0.98582 -0.19035 -0.46205 0.098452 0.05451 -0.091528 -
0.54824 0.5623 0.55934 -0.035081 -1.115 1.0395 0.19568 0.13382 0.19897 0.91299 -0.22714 0.35969 -
0.37996 0.63541 -0.69843 -0.88876 -0.68729 0.031976 1.1844 -0.080089 0.35838 -0.25347 -0.38321
0.058399]

Neural Network II

[-1.8963 -0.28861 0.14693 0.52188 0.0021163 0.45367 -0.013661 1.0718 0.79166 0.12273 -1.8375 -
0.59678 0.28752 -1.3048 -0.47042 -0.05793 -0.95272 0.015503 -0.91325 1.3127 -0.27367 0.37804
0.10196 0.70541 0.14261 -0.6733 0.87104 0.77295 0.53057 -2.6086 -0.16872 0.42282 -0.68744
0.070333 -0.11818 -0.26445 1.035 0.71876 0.52489 -0.3909;

-0.6103 1.4528 0.46415 0.19778 -0.038254 0.78971 1.9252 0.94018 -1.231 -1.161 -0.25352 -0.66053 -
2.79 -0.17144 -0.20007 0.74797 -1.8342 -1.4451 -1.2396 1.7816 0.84104 -1.2499 1.0152 -1.4108
0.016862 -1.0039 2.0336 -0.17936 1.7454 -0.27762 1.1661 0.21102 -0.68995 -2.4038 0.93152 0.77447
0.99105 0.095405 0.93143 -0.5318;

-1.5563 0.24193 -0.40451 -0.63966 -0.55912 0.65793 -1.8159 0.22194 -0.56065 1.9859 -1.2999 -
0.96695 1.1348 -1.8683 -2.2907 1.1997 1.249 -1.4883 -1.4148 -0.094304 1.3466 1.373 0.42275 0.60905
0.73467 -1.0269 0.71054 -0.34894 -1.3968 -0.71592 2.6769 1.0763 -0.15124 0.018598 -0.51099 0.71413
1.1453 0.70292 1.4196 -0.34193;

-2.2961 -3.2184 -2.3318 -0.42701 -3.3586 0.41123 -1.4978 0.89366 1.7587 -4.8844 0.17496 0.099853 -
3.0581 -1.3049 1.3843 1.8752 0.24377 0.036169 -1.0718 2.1406 -2.4268 -0.51735 0.2829 1.411 -2.0611 -
0.48511 0.28103 1.1041 0.37915 -2.1335 -1.47 -0.32884 -1.4898 0.9789 -0.55056 -1.2076 0.24605 -
0.038974 0.3144 -1.5631;

4.1161 0.33019 1.4321 1.7449 2.4248 0.67933 2.0447 0.68722 -0.38552 2.4549 -0.30611 1.7623 3.2181
2.4129 0.34921 -2.7354 0.68459 0.81391 2.3231 -3.3254 -1.3536 1.3529 -1.3844 -0.20604 1.9518 -
0.48443 -2.2806 -0.15088 0.74373 3.1507 -1.8506 0.17068 0.46448 2.2644 -1.1119 1.2558 0.017139
0.63045 0.26262 2.0037]

Appendix VII

Date Published	Reported Vulnerabilities from Security Focus	Butraq ID
Aug 06 2002	Microsoft Windows Window Message Subsystem Design Error Vulnerability	5408
May 25 2005	DavFS2 Failure To Enforce UNIX Filesystem Permissions Design Error Vulnerability	13770
Apr 14 2005	Opera SSL Security Feature Design Error Vulnerability	13176

Jul 01 2005	RaXnet Cacti Config.PHP Design Error Vulnerability	14130
Jan 11 2005	Bottomline Technologies WebSeries Design Error Vulnerabilities	12231
Dec 23 2004	Linux Security Modules Process Capabilities Design Error Vulnerability	12093
Jul 29 2002	Multiple Browser Vendor Same Origin Policy Design Error Vulnerability	5346
May 03 2005	PostgreSQL TSearch2 Design Error Vulnerability	13475
Oct 03 2006	IBM Client Security Password Manager Design Error Vulnerability	20308
Aug 23 2004	SUPHP Design Flaw Local Privilege Escalation Weakness	112020
Apr 30 2004	Web Wiz Forum Multiple Vulnerabilities	10255
Apr 20 2003	Microsoft Windows NTFS Failure To Initialize File Block Vulnerability	7386
Mar 08 2002	Check Point FW-1 SecuClient/SecuRemote Client Design Vulnerability	4253
May 10 2001	NetProwler Password Facilities Weak Design Vulnerability	2727
Mar 02 2005	PHP Glob Function Local Information Disclosure Vulnerability	12701
Mar 22 2004	PHP-Nuke MS-Analysis Module Multiple Remote Path Disclosure Vulnerabilities	9946
Feb 20 2003	Multiple Vendor ATM Hardware Security Module PIN Generation/Verification Vulnerability	6901
Sep 06 2005	MAXdev MD-Pro Arbitrary Remote File Upload	14750
Dec 06 2001	Multiple Personal Firewall Vendor Outbound Packet Bypass Vulnerability	3647
Nov 01 2005	IOFTPD Username Enumeration Vulnerability	15253

Aug 27 2004	MeindISOFT Cute PHP Library cphplib Input Validation Vulnerabilities	11062
Aug 05 2004	Libpng Graphics Library Unspecified Remote Buffer Overflow Vulnerability	10872
Jun 14 2004	Linux Kernel Floating Point Exception Handler Local Denial Of Service Vulnerability	10538
Apr 23 2004	Zonet Wireless Router NAT Implementation Design Flaw Vulnerability	10225
Jul 12 2001	ArGoSoft FTP Server Weak Password Encryption Vulnerability	3029
Nov 28 2005	Microsoft Windows SynAttackProtect Predictable Hash Remote Denial of Service Vulnerability	15613
Aug 29 2005	BFCCommand & Control Server Manager Multiple Remote Vulnerabilities	14690
Feb 18 2005	Tarantella Enterprise/Secure Global Desktop Remote Information Disclosure Vulnerability	12591
Dec 15 2004	Roxio Toast TDIXSupport Local Privilege Escalation Vulnerability	11940
Nov 08 2004	Sun Java Runtime Environment InitialDirContext Remote Denial Of Service Vulnerability	11619
Nov 01 2004	Linux Kernel IPTables Initialization Failure Vulnerability	11570
Dec 20 2002	Multiple Temporary File Monitoring Utility Vendor Stopped Process Vulnerabilities	6451
Feb 18 2002	Cigital ITS4 Software Security Tool Weakness	4120
Aug 30 2000	Stalkerlab's Mailers 1.1.2 CGI Mail Spoofing Vulnerability	1623
Nov 16 2005	Multiple Vendor IpCommandLine Application Path Vulnerability	15448
Nov 16 2005	Counterpane Password Safe Insecure Encryption Vulnerability	15455
Nov 03 2005	F-Prot Antivirus ZIP Attachment Version Scan Evasion Vulnerability	15293

Oct 19 2005	Yiff-Server File Permission Bypass Weakness	14150
Sep 05 2005	Microsoft Windows Keyboard Event Privilege Escalation Weakness	14743
Jul 29 2005	Gopher Insecure Temporary File Creation Vulnerability	14420
Jul 26 2005	IBM Lotus Domino Password Encryption Weakness	14389
Jul 23 2005	RealChat User Impersonation Vulnerability	14358
Jul 18 2005	MRV Communications In-Reach Console Servers Access Control Bypass Vulnerability	14300
Jun 06 2005	LutelWall Multiple Insecure File Creation Vulnerabilities	13863
May 25 2005	GNU SHTool Insecure Temporary File Deletion Vulnerability	13767
May 25 2005	xMySQLadmin Insecure Temporary File Creation Vulnerability	13913
May 04 2005	NetWin DMail DList Remote Authentication Bypass Vulnerability	13497
Apr 29 2005	RedHat Enterprise Linux Native POSIX Threading Library Local Information Disclosure Vulnerability	13444
Apr 28 2005	MyPHP Forum Post.PHP Username Spoofing Vulnerability	13429
Apr 28 2005	MyPHP Forum Privmsg.PHP Username Spoofing Vulnerability	13430
Apr 12 2005	FreeBSD PortUpgrade Local Insecure Temporary File Handling Vulnerability	13106
Apr 07 2005	Macromedia ColdFusion MX Updater Remote File Disclosure Vulnerability	13060
Apr 06 2005	Vixie Cron Crontab File Disclosure Vulnerability	13024
Mar 30 2005	Kerio Personal Firewall Local Network Access Restriction Bypass Vulnerability	12946

Mar 16 2005	Woodstone Servers Alive Local Privilege Escalation Vulnerability	12822
Mar 14 2005	Wine Local Insecure File Creation Vulnerability	12791
Mar 09 2005	KDE Konqueror Remote Download Dialog Box Source URI Spoofing Vulnerability	12769
Feb 28 2005	Mitel 3300 Integrated Communications Platform Web Interface Authentication Bypass Vulnerability	12682
Feb 21 2005	Sun Solaris KCMS_Configure Arbitrary File Corruption Vulnerability	12605
Jan 25 2005	Bribble Unspecified Remote Authentication Bypass Vulnerability	12361
Jan 25 2005	Libdbi-perl Unspecified Insecure Temporary File Creation Vulnerability	12360
Jan 21 2005	Ghostscript Multiple Local Insecure Temporary File Creation Vulnerabilities	12327
Jan 14 2005	SGI InPerson Local Privilege Escalation Vulnerability	12259
Dec 23 2004	LPRNG LPRNG_CERTS.SH Local Insecure Temporary File Creation Vulnerability	12088
Dec 23 2004	Docbook-To-Man Insecure Temporary File Creation Vulnerability	12087
Dec 21 2004	Rosiello Security RPF Multiple Remote And Local Vulnerabilities	12073
Dec 14 2004	Sun Java System Web And Application Server Remote Session Disclosure Vulnerability	11918
Dec 11 2004	Opera Web Browser Download Dialogue Box File Name Spoofing Vulnerability	11883
Dec 02 2004	FreeBSD Linux ProcFS Local Kernel Denial Of Service And Information Disclosure Vulnerability	11789
Nov 22 2004	Citrix MetaFrame Presentation Server Client Debugging Utility Information Disclosure Vulnerability	11720
Nov 20 2004	Computer Associates eTrust EZAntivirus User Interface Local Authentication Bypass Vulnerability	11717

Nov 19 2004	Opera Web Browser Java Implementation Multiple Remote Vulnerabilities	11712
Nov 15 2004	Fcron FCronTab/FCronSighUp Multiple Local Vulnerabilities	11684
Nov 12 2004	OpenSkat Weak Encryption Key Generation Vulnerability	11667
Nov 03 2004	TIPS MailPost Remote Debug Mode Information Disclosure Vulnerability	11595
Nov 02 2004	Minihttp Forum Web Server Plain Text Password Storage Vulnerability	11585
Oct 19 2004	Sun Solaris LDAP RBAC Local Privilege Escalation Vulnerability	11459
Oct 18 2004	Proland Software Protector Plus AntiVirus MS-DOS Name Scan Evasion Vulnerability	11451
Oct 18 2004	Gnofract 4D Remote Script Code Execution Vulnerability	11445
Oct 18 2004	FIL Security Laboratory Twister Anti-TrojanVirus MS-DOS Name Scan Evasion Vulnerability	11453
Oct 12 2004	Microsoft Window Management API Local Privilege Escalation Vulnerability	11378
Oct 12 2004	Adobe Acrobat Reader Remote Access Validation	11386
Sep 15 2004	McAfee VirusScan System Scan Local Privilege Escalation Vulnerability	11181
Sep 28 2004	Vignette Application Portal Remote Information Disclosure Vulnerability	11284
Sep 16 2004	Microsoft Internet Explorer User Security Confirmation Bypass Vulnerability	11200
Sep 15 2004	McAfee VirusScan System Scan Local Privilege Escalation Vulnerability	11181
Sep 13 2004	Lexar JumpDrive Secure USB Flash Drive Insecure Password Storage Vulnerability	11162
Aug 31 2004	D-Link Securicam Network DCS-900 Internet Camera Remote Configuration Vulnerability	11072

Aug 26 2004	Webroot Software Window Washer Data Exposure Vulnerability	11054
Aug 04 2004	DGen Emulator Symbolic Link Vulnerability	10855
Aug 04 2004	YaST2 Utility Library File Verification Shell Code Injection Vulnerability	10867
Jun 24 2004	ZaireWeb Solutions Newsletter ZWS Administrative Interface Authentication Bypass Vulnerability	10605
Jun 16 2004	Check Point Firewall-1 Internet Key Exchange Information Disclosure Vulnerability	10558
Jun 14 2004	Immunix StackGuard Canary Corruption Handler Evasion Vulnerability	10535
Jun 09 2004	Symantec Gateway Security 360R Wireless VPN Bypass Weakness	10502
Jun 08 2004	U.S. Robotics Broadband Router 8003 Administration Web Interface Insecure Password Vulnerability	10490
May 27 2004	PHP Input/Output Wrapper Remote Include Function Command Execution Weakness	10427
May 17 2004	Microsoft Outlook 2003 Media File Script Execution Vulnerability	10369
May 13 2004	Multiple Vendor IEEE 802.11 Protocol Remote Denial Of Service Vulnerability	10342
May 12 2004	Linux Kernel Serial Driver Proc File Information Disclosure Vulnerability	10330
May 04 2004	IPMenu Log File Symbolic Link Vulnerability	10269
Apr 20 2004	Cisco Internet Operating System SNMP Message Processing Denial Of Service Vulnerability	10186
Apr 19 2004	SSMTP Mail Transfer Agent Symbolic Link Vulnerability	10171
Apr 15 2004	Linux Kernel EXT3 File System Information Leakage Vulnerability	10152
Apr 13 2004	Microsoft Windows Object Identity Network Communication Vulnerability	10121

Apr 13 2004	BEA WebLogic Local Password Disclosure Vulnerability	10133
Apr 07 2004	Intel LAN Management Server Setup Utilities Configuration Vulnerability	10068
Apr 02 2004	Macromedia Dreamweaver Remote User Database Access Vulnerability	10036
Mar 26 2004	AIX Invscountd Symbolic Link Vulnerability	9982
Mar 17 2004	Belchior Foundry VCard Authentication Bypass Vulnerability	9910
Mar 15 2004	VocalTec VGW4/8 Telephony Gateway Remote Authentication Bypass Vulnerability	9876
Mar 09 2004	IBM AIX RC.BOOT Local Insecure Temporary File Creation Vulnerability	12992
Feb 25 2004	Mozilla Browser Zombie Document Cross-Site Scripting Vulnerability	9747
May 24 2002	LocalWEB2000 File Disclosure Vulnerability	4820
Jan 11 2001	Microsoft Web Client Extender NTLM Authentication Vulnerability	2199
Nov 18 2000	NetcPlus SmartServer3 Weak Encryption	1962
Nov 18 2000	NetcPlus BrowseGate Weak Encryption Vulnerability	1964
Sep 01 2000	QNX Voyager Webserver Multiple Vulnerabilities	1648
Aug 19 2000	Gnome-Lokkit Firewall Package Port Visibility Vulnerability	1590
Apr 15 2000	QNX crypt() Vulnerability	1114
Mar 22 2000	Multiple Linux Vendor gpm Setgid Vulnerability	1069
Feb 28 2000	OpenSSL Unseeded Random Number Generator Vulnerability	3187

Nov 25 2004	Sun Java Applet Invocation Version Specification Weakness	11757
Jul 20 2000	Microsoft Outlook Express Persistent Mail-Browser Link Vulnerability	1502
Dec 23 2005	SCPOnly Multiple Local Vulnerabilities	16051
Dec 20 2005	MetaDot Portal Server Site_Mgr Group Privilege Escalation Vulnerability	15975
Nov 15 2005	Apple iTunes 6 For Windows Arbitrary Local Code Execution Vulnerability	15446
Sep 29 2005	Macromedia Breeze Plaintext Password Storage Weakness	14975
Sep 26 2005	SecureW2 Insecure Pre-Master Secret Generation Vulnerability	14945
Sep 14 2005	LineControl Java Client Local Password Disclosure Vulnerability	14830
Sep 12 2005	Mark D. Roth PAM_Per_User Authentication Bypass Vulnerability	14813
Sep 07 2005	CSystems WebArchiveX ActiveX Component Arbitrary File Read and Write Vulnerabilities	14760
Jul 20 2005	Greasemonkey Multiple Remote Information Disclosure Vulnerabilities	14336
Jul 18 2005	EKG Insecure Temporary File Creation Vulnerability	14307
Jun 20 2005	Cisco VPN Concentrator Groupname Enumeration Weakness	13992
May 26 2005	Gentoo Webapp-Config Insecure File Creation Vulnerability	13780
May 23 2005	Gibraltar Firewall Antivirus Scan Evasion Vulnerability	13713
May 17 2005	MySQL mysql_install_db Insecure Temporary File Creation Vulnerability	13360
May 02 2005	Apple Mac OS X Default Pseudo-Terminal Permission Vulnerability	13467

May 02 2005	ARPUS Ce/Ceterm Insecure Temporary File Creation Vulnerability	13465
Apr 26 2005	Rootkit Hunter Local Insecure Temporary File Creation Vulnerability	13399
Apr 16 2005	Webmin And Usermin Configuration File Unauthorized Access Vulnerability	13205
Apr 11 2005	RSnapshot Local File Permission Manipulation Vulnerability	13095
Apr 04 2005	GNU Sharutils Unshar Local Insecure Temporary File Creation Vulnerability	12981
Apr 04 2005	Remstats Local Insecure Temporary File Creation Vulnerability	12979
Mar 23 2005	Mathopd Dump Files Local Insecure File Creation Vulnerability	12882
Mar 11 2005	Xerox WorkCentre Multiple Page Fax Information Disclosure Vulnerability	12787
Feb 18 2005	Yahoo! Messenger Download Dialogue Box File Name Spoofing Vulnerability	12587
Feb 14 2005	Debian Toolchain-Source Multiple Insecure Temporary File Creation Vulnerabilities	12540
Feb 11 2005	OpenPGP Cipher Feedback Mode Chosen-Ciphertext Partial Plaintext Retrieval Vulnerability	12529
Jan 27 2005	F2C Multiple Local Insecure Temporary File Creation Vulnerabilities	12380
Jan 26 2005	Apple Mail EMail Message ID Header Information Disclosure Vulnerability	12366
Dec 31 2004	ArGoSoft FTP Server Remote User Enumeration Vulnerability	12139
Dec 23 2004	Debian Tetex-Bin Xdvizilla Insecure Temporary File Creation Vulnerability	12100
Dec 22 2004	Debian Debmake Local Insecure Temporary File Creation Vulnerability	12078
Dec 21 2004	Webroot Software Spy Sweeper Enterprise Local Privilege Escalation Vulnerability	12065

Dec 21 2004	Webroot Software My Firewall Plus Local Privilege Escalation Vulnerability	12064
Dec 20 2004	GNU Troff (Groff) Insecure Temporary File Creation Vulnerabilities	12058
Dec 15 2004	MoniWiki Remote Server-Side Script Execution Vulnerability	11951
Dec 10 2004	Kerio WinRoute Firewall Multiple Unspecified Remote Vulnerabilities	11870
Dec 07 2004	Gentoo MirrorSelect Local Insecure File Creation Vulnerability	11835
Nov 23 2004	Van Dyke SecureCRT Remote Command Execution Vulnerability	11731
Nov 17 2004	Cscope Insecure Temporary File Creation Vulnerabilities	11697
Nov 12 2004	GratiSoft Sudo Restricted Command Execution Bypass Vulnerability	11668
Nov 11 2004	Davfs2 Insecure Temporary File Creation Vulnerability	11661
Nov 10 2004	Mozilla Firefox Download Dialogue Box File Name Spoofing Vulnerability	11643
Nov 02 2004	Haserl Local Environment Variable Manipulation Vulnerability	11579
Oct 27 2004	Apple Remote Desktop Administrator Privilege Escalation Vulnerability	11554
Oct 18 2004	H+BEDV AntiVir MS-DOS Name Scan Evasion Vulnerability	11444
Oct 05 2004	Symantec Norton AntiVirus MS-DOS Name Scan Evasion Vulnerability	11328
Sep 30 2004	Perl Unspecified Insecure Temporary File Creation Vulnerability	11294
Sep 30 2004	OpenSSL DER_CHOP Insecure Temporary File Creation Vulnerability	11293
Sep 30 2004	NetaTalk Unspecified Insecure Temporary File Creation Vulnerability	11292

Sep 30 2004	MySQL Unspecified Insecure Temporary File Creation Vulnerability	11291
Sep 30 2004	Trustix LVM Utilities Unspecified Insecure Temporary File Creation Vulnerability	11290
Sep 30 2004	MIT Kerberos 5 SEND-PR.SH Insecure Temporary File Creation Vulnerability	11289
Sep 30 2004	GNU GZip Unspecified Insecure Temporary File Creation Vulnerability	11288
Sep 30 2004	GNU Troff (Groff) Groffer Script Insecure Temporary File Creation Vulnerability	11287
Sep 30 2004	GNU GLibC Insecure Temporary File Creation Vulnerability	11286
Sep 30 2004	GhostScript Insecure Temporary File Creation Vulnerability	11285
Sep 30 2004	GNU GetText Unspecified Insecure Temporary File Creation Vulnerability	11282
Sep 30 2004	PostgreSQL Insecure Temporary File Creation Vulnerability	11295
Sep 23 2004	Motorola WR850G Wireless Router Remote Authentication Bypass Vulnerability	11241
Sep 22 2004	Sophos Anti-Virus Reserved MS-DOS Name Scan Evasion Vulnerability	11236
Sep 21 2004	Symantec ON Command CCM Remote Database Default Password Vulnerability	11225
Sep 15 2004	Multiple Browser Cross-Domain Cookie Injection Vulnerability	11186
Aug 17 2004	Gallery Remote Server-Side Script Execution Vulnerability	10968
Aug 05 2004	Mozilla Browser Non-FQDN SSL Certificate Spoofing Vulnerability	10876
Aug 04 2004	LILO gfxboot Plaintext Password Display Vulnerability	10866
Aug 04 2004	Linux Kernel File 64-Bit Offset Pointer Handling Kernel Memory Disclosure Vulnerability	10852

Aug 02 2004	Gnu Transport Layer Security Library X.509 Certificate Verification Denial Of Service Vulnerability	10839
Jul 21 2004	Serena TeamTrack Remote Authentication Bypass Vulnerability	10770
Jul 20 2004	Sysinternals PsTools Remote Unauthorized Access Vulnerability	10759
Jul 13 2004	4D WebStar Symbolic Link Vulnerability	10714
Jun 17 2004	Sun Solaris Patches 112908-12 And 115168-03 Clear Text Password Logging Vulnerability	10606
Apr 22 2004	Xine And Xine-Lib Multiple Remote File Overwrite Vulnerabilities	10193
Apr 21 2004	BEA WebLogic Server And WebLogic Express Configuration Log Files Plain Text Password	10188
Apr 19 2004	Softwin BitDefender AvxScanOnlineCtrl COM Object Information Disclosure Vulnerability	10175
Apr 19 2004	Softwin BitDefender AvxScanOnlineCtrl COM Object Remote File Upload And Execution Vulnerability	10174
Mar 30 2004	LinBit Technologies LinBox Plain Text Password Storage Weakness	10011
Mar 23 2004	Mythic Entertainment Dark Age of Camelot Encryption Key Signing Vulnerability	9960
Mar 19 2004	Samba SMBPrint Sample Script Insecure Temporary File Handling Symbolic Link Vulnerability	9926
Mar 12 2004	Macromedia Studio MX 2004 /Contribute 2 Local Privilege Escalation Vulnerability	9862
Mar 12 2004	XInterceptTalk XITalk Privilege Escalation Vulnerability	9851
Mar 09 2004	F-Secure SSH Server Password Authentication Policy Evasion Vulnerability	9824
Feb 25 2004	MTools MFormat Privilege Escalation Vulnerability	9746
Sep 07 2003	Microsoft ISA Server HTTP Authentication Scheme Vulnerability	10481

Mar 05 2003	CatDoc XLSView Local Insecure Temporary File Creation Vulnerability	11560
Oct 09 2002	Microsoft Windows NetDDE Privilege Escalation Vulnerability	5927
Dec 28 2002	ShadowJAAS Command Line Password Disclosure Vulnerability	6498
Feb 14 2002	Microsoft Visual C++ 7/Visual C++.Net Buffer Overflow Protection Weakness	4108
Feb 09 2002	Adobe PhotoDeluxe Java Execution Vulnerability	4106
Dec 18 2001	GTK Shared Memory Permissions Vulnerability	3705
Dec 01 2000	Microsoft Internet Explorer 'INPUT TYPE=FILE' Vulnerability	2045
Jan 16 2006	Albatross Remote Arbitrary Code Execution Vulnerability	16252
Sep 01 2005	PolyGen Local Denial of Service Vulnerability	14722
Mar 30 2005	GDK-Pixbuf BMP Image Processing Double Free Remote Denial of Service Vulnerability	12950
Jul 26 2002	T. Hauck Jana Server FTP Server PASV Mode Port Exhaustion Denial Of Service Vulnerability	5325
Jan 19 2006	Ecartis PantoMIME Arbitrary Attachment Upload Vulnerability	16317
Jan 12 2006	Microsoft Visual Studio UserControl Remote Code Execution Vulnerability	16225
Dec 23 2005	RSSH RSSH_CHROOT_HELPER Local Privilege Escalation Vulnerability	16050
Dec 22 2005	WebWasher Malicious Script Filter Bypass Vulnerability	16047
Dec 21 2005	Cisco Downloadable RADIUS Policies Information Disclosure Vulnerability	16025
Dec 20 2005	Clearswift MIMESweeper For Web Executable File Bypass Vulnerability	15982

Dec 13 2005	Microsoft Internet Explorer COM Object Instantiation Memory Corruption Vulnerability	15827
Dec 13 2005	Opera Web Browser Download Dialog Manipulation File Execution Vulnerability	15835
Dec 08 2005	PGP Desktop Wipe Free Space Assistant Improper Disk Wipe Vulnerability	15784
Dec 06 2005	Sun Java System Application Server Reverse SSL Proxy Plug-in Man In The Middle Vulnerability	15728
Nov 21 2005	IBM WebSphere Application Server for z/OS Double Free Denial of Service Vulnerability	15522
Nov 07 2005	Zone Labs Zone Alarm Advance Program Control Bypass Weakness	15347
Oct 29 2005	PHP Advanced Transfer Manager Remote Unauthorized Access Vulnerability	15237
Oct 28 2005	Rockliffe MailSite Express Arbitrary Script File Upload Vulnerability	15230
Oct 19 2005	HP-UX FTP Server Directory Listing Vulnerability	15138
Oct 10 2005	SGI IRIX Runpriv Local Privilege Escalation Vulnerability	15055
Oct 06 2005	Planet Technology FGSW-2402RS Switch Backdoor Password Reset Vulnerability	15014
Oct 04 2005	Microsoft Windows Wireless Zero Configuration Service Information Disclosure Vulnerability	15008
Sep 22 2005	Linux Kernel 64-Bit SMP Routing_ioctl() Local Denial of Service Vulnerability	14902
Sep 17 2005	Py2Play Object Unpickling Remote Python Code Execution Vulnerability	14864
Sep 17 2005	Tofu Object Unpickling Remote Python Code Execution Vulnerability	14865
Sep 13 2005	Linksys WRT54G Wireless Router Multiple Remote Vulnerabilities	14822
Sep 13 2005	Apple Mac OS X Untrusted Java Applet Privilege Escalation Vulnerability	14826

Sep 02 2005	FileZilla FTP Client Hard-Coded Cipher Key Vulnerability	14730
Aug 31 2005	Symantec LiveUpdate Client Local Information Disclosure Vulnerability	14708
Aug 30 2005	Maildrop Lockmail Local Privilege Escalation Vulnerability	14696
Aug 22 2005	RunCMS Arbitrary Variable Overwrite Vulnerability	14634
Aug 03 2005	Symantec Norton GoBack Local Authentication Bypass Vulnerability	14461
Aug 02 2005	nCipher CHIL Random Cache Leakage Vulnerability	14452
Jul 27 2005	Linux Kernel SYS_GET_THREAD_AREA Information Disclosure Vulnerability	15527
Jul 26 2005	Linux Kernel NAT Handling Memory Corruption Denial of Service Vulnerability	15531
Jul 26 2005	IBM Lotus Domino WebMail Information Disclosure Vulnerability	14388
Jul 20 2005	Oray PeanutHull Local Privilege Escalation Vulnerability	14330
Jul 19 2005	Multiple Browser Weak Authentication Mechanism Vulnerability	14325
Jul 06 2005	eRoom Plug-In Insecure File Download Handling Vulnerability	14176
Jul 01 2005	OpenLDAP TLS Plaintext Password Vulnerability	14125
Jul 01 2005	PADL Software PAM_LDAP TLS Plaintext Password Vulnerability	14126
May 28 2005	Invision Power Board Privilege Escalation Vulnerability	13797
Apr 24 2005	ACS Blog Administrative Access Authentication Bypass Vulnerability	13346
Mar 17 2005	ThePoolClub IPool/ISnooker Insecure Local Credential Storage Vulnerability	12830

Mar 10 2005	Multiple Vendor Antivirus Products Malformed ZIP Attachment Scan Evasion Vulnerability	12771
Feb 24 2005	Cyclades AlterPath Manager Multiple Remote Vulnerabilities	12649
Dec 21 2004	PHPAuction Administrative Interface Authentication Bypass Vulnerability	12069
Dec 14 2004	Multiple Kerio Products Universal Secret Key Storage Vulnerability	11930
Nov 12 2004	Alcatel Speed Touch Pro With Firewall ADSL Router DNS Poisoning Vulnerability	11664
Oct 18 2004	Best Software SalesLogix Multiple Remote Vulnerabilities	11450
Oct 08 2004	Nathaniel Bray Yeemp File Transfer Public Key Verification Bypass Vulnerability	11353
Aug 10 2004	Sygate Secure Enterprise Enforcer Unauthenticated Broadcast Request Bypass Vulnerability	10908
May 26 2004	FreeBSD Msync(2) System Call Buffer Cache Implementation Vulnerability	10416
Oct 02 2003	Microsoft Windows PostThreadMessage() Arbitrary Process Killing Vulnerability	8747
Sep 10 2003	CacheFlow CacheOS HTTP HOST Proxy Vulnerability	8584
Aug 11 2003	FreeBSD Ptrace/SPIgot Insufficient Signal Verification Denial of Service Vulnerability	8387
Jun 19 2003	Power Server FTP Addon Failure To Authenticate Vulnerability	7986
Jun 13 2003	Cistron RADIUS Remote Signed NAS-Port Number Expansion Memory Corruption Vulnerability	7892
Apr 30 2003	ScriptLogic RunAdmin Service Administrative Access Vulnerability	7477
Mar 17 2003	Multiple Cryptographic Weaknesses in Kerberos 4 Protocol	7113
Dec 26 2002	Microsoft Windows File Protection Code-Signing Verification Weakness	6482

Nov 15 2002	TightVNC Server Authentication Cookie Predictability Vulnerability	6905
Oct 21 2002	Multiple Firewall Vendor Packet Flood State Table Filling Vulnerability	6023
Aug 17 2002	Microsoft Internet Explorer XML Datasource Applet File Disclosure Vulnerability	5490
Jul 03 2002	Sun SunPCi II VNC Software Password Disclosure Vulnerability	5146
May 17 2002	GRSecurity Linux Kernel Memory Protection Weakness	4762
Apr 24 2002	Multiple Stack Protection Scheme Function Argument Overwrite Weakness	4586
Apr 16 2002	Pipemail/Mailman Insecure Archives Permissions Vulnerability	4538
Feb 18 2002	Compaq Tru64 SNMP Agent Denial Of Service Vulnerability	4140
Feb 14 2002	W3C CSS :visited Pseudo-Class Information Disclosure Vulnerability	4136
Jan 21 2002	GNU Enscript Insecure Temporary File Creation Vulnerability	3920
Jan 06 2002	Linksys DSL Router SNMP Trap System Arbitrary Sending Vulnerability	3795
Dec 17 2001	Microsoft Windows XP Unauthorized Hotkey Program Execution Vulnerability	3703
Dec 07 2001	Microsoft Windows File Locking DoS Vulnerability	3654
Aug 02 2001	Identix BioLogon Client Biometric Authentication Bypass Vulnerability	3140
May 15 2001	Logitech Wireless Peripheral Device Man in the Middle Vulnerability	2738
Apr 02 2001	Lucent Orinoco Closed Network Unauthorized Access Vulnerability	2538
Mar 20 2001	OpenPGP Private Key Attack Vulnerability	2673

Sep 02 1999	IEEE 802.1q Unauthorized VLAN Traversal Weakness	615
Jan 17 2006	Mozilla Thunderbird File Attachment Spoofing Vulnerability	16271
Jan 03 2006	Gentoo Pinentry Local Privilege Escalation Vulnerability	16120
Dec 30 2005	Gentoo Linux XnView Insecure RPATH Vulnerability	16087
Dec 22 2005	MediaWiki Inline Style Attribute Security Check Bypass Vulnerability	16032
Dec 15 2005	Multiple Vendor Wireless Access Points Static WEP Key Authentication Bypass Vulnerability	16068
Dec 07 2005	Sun Solaris Sun Update Connection Web Proxy Password Disclosure Vulnerability	15772
Nov 15 2005	Macromedia Contribute Publishing Server Insecure Shared Connection Key Encryption Weakness	15438
Oct 07 2005	SuSE YaST Package Repositories Insecure Permissions Vulnerability	15026
Oct 07 2005	Oracle HTML DB Plaintext Password Storage Vulnerability	15033
Sep 19 2005	Sybari Antigen for Exchange/SMTP Attachment Rule Bypass Vulnerability	14875
Sep 06 2005	Gentoo Net-SNMP Local Privilege Escalation Vulnerability	14845
Jul 29 2005	Novell eDirectory NMAS Authentication Bypass Vulnerability	14419
Jul 28 2005	Opera Web Browser Content-Disposition Header Download Dialog File Extension Spoofing Vulnerability	14402
Jul 27 2005	BSD IPsec Session AES-XCBC-MAC Authentication Constant Key Usage Vulnerability	14394
Jul 19 2005	Apple Mac OS X AirPort Card Automatic Network Association Vulnerability	14321
Jul 15 2005	Macromedia JRun Unauthorized Session Access Vulnerability	14271

Jul 14 2005	BitDefender Antivirus & Antispam for Linux and FreeBSD Mail Servers Scan Evasion	14262
Jul 11 2005	Backup Manager Insecure Temporary File Creation Vulnerability	14210
Jul 07 2005	PHPSlash Arbitrary Account Privilege Escalation Vulnerability	14189
Jun 29 2005	FreeBSD TCP Stack Established Connection Denial of Service Vulnerability	14104
Jun 27 2005	Adobe Acrobat/Adobe Reader Safari Frameworks Folder Permission Escalation Vulnerability	14075
Jun 01 2005	I-Man File Attachments Remote Arbitrary PHP Script Execution Vulnerability	13831
May 17 2005	IgnitionServer Entry Deletion Access Validation Checking Vulnerability	13654
Mar 29 2005	Linux Kernel EXT2 File System Information Leak Vulnerability	12932
Feb 14 2005	Microsoft Internet Explorer Mouse Event URI Status Bar Obfuscation Weakness	12541
Feb 04 2005	Postfix IPv6 Unauthorized Mail Relay Vulnerability	12445
Jan 28 2005	WebWasher Classic HTTP CONNECT Unauthorized Access Weakness	12394
Jan 27 2005	Ingate Firewall Persistent PPTP Tunnel Vulnerability	12383
Dec 07 2004	MD5 Message Digest Algorithm Hash Collision Weakness	11849
Oct 01 2004	Sun Solaris Gzip File Permission Modification Vulnerability	11318
Sep 17 2004	Apple iChat Remote Link Application Execution Vulnerability	11207
May 13 2004	Opera Web Browser Address Bar Spoofing Weakness	10337
Apr 30 2004	Sun Solaris Patch Information Disclosure Vulnerability	10261

Mar 12 2004	Sun Solaris Patch Unexpected Security Weakness	9852
Nov 05 2003	Microsoft Internet Explorer Double Slash Cache Zone Bypass Vulnerability	8980
Jun 11 2003	Ethereal TVB_GET_NSTRINGZ0() Memory Handling Vulnerability	7883
May 14 2003	Linux Kernel Route Cache Entry Remote Denial Of Service Vulnerability	7601
May 05 2003	Mod_Survey SYSBASE Disk Resource Consumption Denial of Service Vulnerability	7498
Apr 29 2003	Microsoft Log Sink Class ActiveX Control Arbitrary File Creation Vulnerability	12646
Apr 10 2003	Apple MacOS X DropBox Folder Information Disclosure Vulnerability	7324
Dec 05 2002	Akfingerd Remote Denial Of Service Vulnerability	6323
Sep 19 2002	Microsoft Virtual Machine Multiple JDBC Vulnerabilities	5478
Sep 11 2002	KDE Konqueror Sub-Frames Script Execution Vulnerability	5689
Jun 17 2002	Mozilla Netscape Navigator Plug-In Path Disclosure Vulnerability	5741
May 29 2002	Core APM File Upload Execution Vulnerability	4922
May 15 2002	Swatch Throttled Event Reporting Vulnerability	4746
Apr 08 2002	Microsoft Office Web Components Local File Read Vulnerability	4453
Jan 31 2002	Microsoft Site Server LDAP Plain Text Password Storage Vulnerability	4000
Dec 12 2001	Util-Linux Script Command Arbitrary File Overwrite Vulnerability	16280
Nov 01 2001	LibDB SNPrintf Buffer Overflow Vulnerability	3497

Oct 05 2001	Symantec Norton Antivirus LiveUpdate Host Verification Vulnerability	3403
Aug 14 2001	Dell Latitude C800 Bios Suspended Session Bypassing Vulnerability	3180
Apr 07 1999	NetworkAppliance NetCache SNMP Default Community String Vulnerability	2807
Sep 30 1996	IETF RADIUS Dictionary Attack Vulnerability	3532
Apr 28 2009	Multiple Symantec Products Alert Management System Console Arbitrary Code Execution Vulnerability	34675
Apr 02 2008	Symantec AutoFix Tool ActiveX Control Remote Share 'launchProcess()' Insecure Method Vulnerability	28509
Mar 23 2007	Sun Java System Directory Server Uninitialized Pointer Remote Memory Corruption Vulnerability	23117
Mar 27 2009	IBM Tivoli Storage Manager Multiple Vulnerabilities	34285
Mar 25 2009	Cisco IOS Secure Copy Remote Privilege Escalation Vulnerability	34247
Dec 29 2003	Microsoft IIS Failure To Log Undocumented TRACK Requests Vulnerability	9313
Dec 17 2008	Mozilla Firefox/Thunderbird/SeaMonkey Multiple Remote Vulnerabilities	32882
Oct 11 2008	Debian chm2pdf Insecure Temporary File Creation Vulnerability	31735

Date Published	www.cigital.com/whitepapers/dl/wp-qandr.pdf	Secunia: ID
Jul 03 2008	Drupal Outline Designer Security Bypass	30936
May 29 2009	SonicWALL Global Security Client Privilege Escalation Vulnerability	35220
Apr 29 2009	Symantec Products Alert Management System 2 Multiple Vulnerabilities	34856
Mar 03 2009	Cisco Unified Communications Manager IP Phone PAB Information Disclosure	34238
Nov 26 2008	Crossday Discuz! Board Multiple Vulnerabilities	32731
Oct 21 2008	Symantec Altiris Deployment Solution Privilege Escalation	31773
Oct 08 2008	Adobe Flash Player "Clickjacking" Security Bypass Vulnerability	32163
Jul 25 2008	RealNetworks RealPlayer Multiple Vulnerabilities	27620
Jun 04 2008	Sun Java System Active Server Pages Multiple Vulnerabilities	30523

Feb 20 2008	Opera Multiple Vulnerabilities	29029
Feb 08 2008	Mozilla Firefox Multiple Vulnerabilities	28758
Nov 15 2007	IBM DB2 Multiple Vulnerabilities and Security Issue	27667
Aug 01 2007	Mac OS X Security Update Fixes Multiple Vulnerabilities	26235
Jul 20 2007	Citrix Access Gateway Multiple Vulnerabilities	26143
Jul 12 2007	Apple QuickTime Multiple Vulnerabilities	26034
Jul 02 2007	Firefox "OnKeyDown" Event Focus Weakness	25904
May 30 2007	Apple QuickTime Java Extension Two Vulnerabilities	25130
May 11 2007	Sun SRS Proxy Core "srsexec" Information Disclosure	25194
May 10 2007	Symantec Products NAVOpts.dll ActiveX Control Security Bypass Vulnerability	25172
May 01 2007	VMware Products Multiple Vulnerabilities	25079
Feb 22 2007	Cisco Secure Services Client Multiple Vulnerabilities	24258
Feb 22 2007	Trend Micro ServerProtect for Linux Web Interface Authentication Bypass	24264
Feb 22 2007	Cisco Unified IP Conference Station / IP Phone Default Accounts	24262
Jan 29 2007	smb4K Multiple Vulnerabilities	23937
Oct 20 2006	Kaspersky Labs Anti-Virus IOCTL Privilege Escalation	22478
Aug 01 2006	MySQL MERGE Table Privilege Revoke Bypass	21259
Jul 10 2006	Flash Player Unspecified Vulnerability and "addRequestHeader()" Bypass	20971
Jun 06 2006	Firefox File Upload Form Keystroke Event Cancel Vulnerability	20442
Apr 25 2006	iOpus Secure Email Attachments Password Usage Security Issue	19771
Apr 24 2006	Symantec Scan Engine Multiple Vulnerabilities	19734
Mar 02 2006	NCP Secure Entry/Enterprise Client Two Vulnerabilities	19082
Feb 20 2006	PHP-Nuke CAPTCHA Bypass Weakness	18936
Jan 23 2006	Tor Hidden Service Disclosure Weakness	18576
Jan 19 2006	Ecartis "pantomime" Functionality Attachment Handling Security Issue	18524
Jan 18 2006	Oracle Products Multiple Vulnerabilities and Security Issues	18493
Jan 11 2006	Symantec Norton SystemWorks Protected Recycle Bin Weakness	18402
Jan03 2006	Cisco Secure Access Control Server Downloadable IP Access Control List Vulnerability	18141
Dec 23 2005	scponly Privilege Escalation and Security Bypass Vulnerabilities	18223
Dec 23 2005	rsch "chroot" Directory Privilege Escalation Vulnerability	18224
Dec 22 2005	Sygate Protection Agent Protection Bypass Vulnerability	18175
Dec 13 2005	Microsoft Internet Explorer Multiple Vulnerabilities	15368
Dec 12 2005	Blackboard Learning and Community Portal Systems Multiple Vulnerabilities	17991
Dec 05 2005	e107 "rate.php" Redirection and Multiple Rating Weakness	17890
Nov 16 2005	PEAR Installer Arbitrary Code Execution Vulnerability	17563
Sep 21 2005	Antigen for Exchange "Antigen forwarded attachment" Filter Bypass	16759