A Neural Network Based Security Tool for Analyzing Software

Adetunji Adebiyi, Johnnes Arreymbi, Chris Imafidon

A Neural Network Based Security Tool for Analyzing Software

Adetunji Adebiyi[1], Johnnes Arreymbi[1] and Chris Imafidon[1]

[1] School of Architecture, Computing and Engineering, University of East London, London, UK
adetunjib@hotmail.com, {J.Arreymbi, C.O.Imafidon}@uel.ac.uk

Abstract. The need to secure software application in today's hostile computer environment cannot be overlooked. The increase in attacks aimed at software directly in the last decade and the demand for more secure software applications has drawn the attention of the software industry into looking for better ways in which software can be developed more securely. To achieve this, it has been suggested that security needs to be integrated into every phase of software development lifecycle (SDLC). In line with this view, security tools are now used during SDLC to integrate security into software applications. Here, we propose a neural network based security tool for analyzing software design for security flaws. Our findings show that the trained neural network was able to match possible attack patterns to design scenarios presented to it. With the information on the attack pattern identified, developers can make informed decision in mitigating risks in their designs.

Keywords: Threat Modeling, Neural Network, Attack Patterns, Security Tools, Secure software design

1 Introduction

The dependence of our society today on software systems demand that they are robust, reliable and secured when they are deployed. Therefore, it is very important that the design of software must be that which makes it to function properly in a hostile computer environment, protecting the information and systems on which it runs with minimal risks. However, as software attacks become more sophisticated especially with the increase of attack tools available, today's security solutions are no longer adequate in providing security[5][18] because many of these attacks circumvent the traditional defenses and target the software directly [2]. By retrofitting security into software after development exacerbate the issue as these may lead to significant change in the architectural design and code of the software which often introduces more flaws and increases the cost of production [4].

Reportedly, 50% of security problems in software products today have been found to be design flaws [8]. In this view, many authors argue 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 development and much more costly when the software has been deployed [4][11][18]. To achieve this, a neural network based

tool that would enable software engineers to evaluate their software design for security flaws by matching attack patterns to software design scenarios presented to it is proposed in this paper. Each attack pattern matched to the design helps the software designer to see areas of vulnerability in the design that needs to be secured. Based on this information, software designers can integrate security capabilities that will mitigate the identified risks in their design before the software is coded.

2 Internet of Things (IoT)

One of the core issues in IoT is the connectivity between social, cloud, mobile and everyday objects. While IoT has lots of benefits, it also increases the attack surface of applications running on the connected devices. Many devices that were never intended to be connected to the internet become exposed to software-based attacks while connected to other devices. Furthermore, as devices connected in IoT will be generating a huge amount of data, the risk of data breach also increases. Therefore, the security of software running these devices connected in IoT cannot be overlooked, This paper contributes in this area by proposing a tool based on neural network that can be used to analyze the software design for security flaws.

3 Current Security Tools for Analyzing Software Design

In recent years, various security tools have been developed to enable software developers who are not security experts to scrutinize their software design and identify security flaws in a similar way as a security expert. Microsoft developed two of these security tools [10] [11]. The first was Threat Analysis and Modeling (TAM) tool. This was developed 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 modeling 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. The second is SDL Threat Modeling Tool. This is a core element in the design phase of Microsoft Security Development Lifecycle which helps software developers to analyze their software designs prior to its implementation. Also, this tool was not developed for security experts but for software developers to aid the creation and analysis of threat models [10]. In contrast to TAM tool, SDL Threat Modeling 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 modeling with this tool focuses on the software and the analysis of its design [17].

While these tools have lots of useful features that enable software developers to do threat modeling 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 Modeling Tool. Thirdly the process of threat modeling can increasingly become complex while using the tools due to factors such as number of developers involved in the threat modeling process, the nature of DFD created and potential stakeholders [1][11].

There is now a range of security tools from open source with similar threat modeling approach like that of Microsoft threat modeling 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 modeling approaches based on standards such as the Risk Analysis Toolkit (based on ISO 1799) which generates security polices from question and answers [15] and other open security tools like the Common Vulnerability Scoring System (CVSS) that is designed to for rating IT vulnerabilities [3].

4 The Neural Network Tool

Previous researches show various ways through which neural network have been used in the area of security. Neural network based applications has been used successfully in the area of network security as intrusion detection systems, misuse detection systems and firewalls [19] [21] [22]. Also in the field of application security, neural network has been proposed to be used as virus detection system [20]. It would be noticed however, that these neural network based applications can only provide a form security after software deployment.

Our 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. Using well known approaches such as DFD and sequence diagrams, software developers are able to abstract information about their software designs needed by the Neural Network tool for matching possible attack patterns. When potential attack patterns are matched against the design, the software developers are able to take the necessary steps in mitigating the security flaw identified. Thus software developers are able to integrate security into their software design during the design phase of SDLC when it is easy and cost effective to resolve security problems.

One of the limitations with some of the current approaches is the difficulty of getting software developers to think like attackers during the threat modeling process as this mindset is not native to them [17]. It has been suggested that software developers can instead look at the attack surface of their software design and think of how to build defenses into their application [17]. Our proposed technique achieve 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 thereby addressing security defenses needed to be put in place.

4.1 The Neural Network Architecture

A three-layered 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 [16]. A back-propagation network has been used in this research because of its simplicity and reasonable speed. The architecture of the back-propagation network is shown in the figure below. This consists of the input layer, the hidden layer and the output layer. Each of the hidden nodes and output nodes apply a tan-sigmoid transfer function (2/(1+exp(-2*n))-1) to the various connection weights. 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 a have better chance of giving a correct output when it is next presented with same input

4.2 Data Collection

Data of attack scenarios from online vulnerability databases such as CVE Details, Security Tracker, Secunia, Security Focus and The Open Source Vulnerability Database were used in this research. From the online vulnerability databases a total of 715 attack scenarios relating to 51 regularly expressed attack patterns by Williams and Gegick [4] were analyzed. This consisted of 260 attack scenarios which were unique in terms of their impact, mode of attack, software component and actors involved in the attack and 455 attack scenarios which are repetition of the same type of exploit in different applications they have been reported in the vulnerability databases. The attacks were analyzed to identify the actors, goals and resources under attack. Once these were identified the attack attributes in Table 1 were used to abstract the data capturing the attack scenario for training the neural network.

4.3 Data Encoding

The training data samples each consist of 12 input units for the neural network. This corresponds to the values of the attributes abstracted from the attack scenarios. The training data was generated from the attack scenarios using the attributes. For instance training data for the attack on webmail (CVE 2003-1192) was generated by looking at the online vulnerability databases to get its details on the attributes we are interested in. This attack corresponds to regularly expressed attack pattern 3. Williams and Gegick [4] describe the attack scenario in this attack pattern as a user submitting an excessively long HTTP GET request to a web server, thereby causing a buffer. This attack pattern is represented as:

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

Table 1. Sample of pre-processed training data from attack scenario

S\N	**Attribute**	**Observed data**	**Value**
1	Attacker	No Access	0
2	Source	External	1
3	Target	Buffer	9
4	Attack Vector	Long Get Request	39
5	Attack Type	Availability	5
6	Input Validation	Partial Validation	2
7	Dependencies	Authentication & Input	6
8	Output	None	0
	Authentication	None	0
10	Access Control	URL Access	2
11	HTTP Security	Input Validation	3
12	Error	None	0

In this example, the data generated from the attack scenario using the attribute list is shown in Table 1. Using the corresponding values for the attributes; the data is then encoded as shown in the Table 1. The second stage of the data processing involves converting the value of the attributes in Table II into ASCII comma delimited format before it is used in training the neural network. For the expected output from the neural network, the data used in training network is derived from the attack pattern which has been identified in each of the attack scenarios. Each attack pattern is given a unique ID which the neural network is expected to produce as an output for each of the input data samples. The output data sample consists of output units corresponding to the attack pattern IDs. For instance, the above sample data on Webmail attack which corresponds to regularly expressed attack pattern 3, the neural network is trained to identify the expected attack pattern as 3.

4.4 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 (51 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. 51 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. 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.

4.5 Result and Discussion

It took the system about one minute to complete the training for each the back-propagation neural network. For the first neural network, the training stopped when the MSE of 0.0016138 was reached at the 26th iteration. The training of the second neural network stopped when the MSE of 0.00012841 was reached at the 435th iteration.

To test the performance of the network, the second data sets were used to test the neural network. It was observed that the trained neural network gave an output as close as possible to the anticipated output. The actual and anticipated outputs are compared in the Table 4. The test samples in which the neural network gave a different output from the predicted output when testing the network includes tests for attack patterns 10, 35, 39, 40 and 52. While looking into the reason behind this, it was seen that the data observed for these attack patterns were not much. With more information on these attack patterns for training the neural network, it is predicted that the network will give a better performance. During the study of the results from the neural networks, it was found that the first neural network had 96.51% correct results while the second neural network had 92% accuracy. The accuracy for both neural networks had an average of 94.1%. Given the accuracy of the neural networks, it shows that neural networks can be used to assess the security in software designs

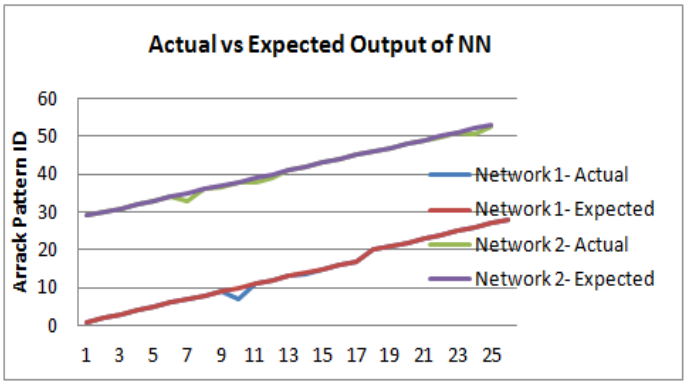

Fig 2. Actual vs. Expected Output of the Neural Network

5 Conclusion

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, the use of the proposed neural networks tool for analyzing software design for security flaws will consolidate the efforts of software developers in identifying areas of security weakness in their software design. By fixing the security flaws in design before coding begins will subsequently lead to the development of more secured software applications. 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.

6 Future Work

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 analyzing software design for security flaws largely depends on the input data capturing the attributes of the software design introduced to it. As this requires a human endeavor, further work is required in this area to ensure that correct input data is retrieved for evaluation. In addition, the neural network needs to be thoroughly tested before it can gain acceptance as a tool for evaluating software design for security flaws. To further improve the performance of the neural network system as a tool for evaluating software design, we are currently looking into the possibility of the system suggesting solutions that can help to prevent the identified attacks. Current research on solutions to software design security flaws gives a good insight in this area. Suggested solutions such as the use security patterns [6] and introduction of security capabilities into design in the SAT [12]

References

1. Berg, B., SDL: Threat Modeling tools vs. Threat Analysis tool,http://www.dib0.nl/code/166-sdl-threat-modeling-tool-vs-threat-analysis-tool
2. Burns, S. F., Threat Modeling: A Process to Ensure Application Security, SANS Institute InfoSec Reading Room, http://www.sans.org/reading_room/whitepapers/securecode/threat-modeling-process-ensure-application- security_1646
3. Common Vulnerability Scoring System (CVSS-SIG), http://www.first.org/cvss
4. Gegick, M. and Williams, L. On the design of more secure software-intensive systems by use of attack patterns', Information and Software Technology, Vol.49, pp381-397, (2006)
5. Keary, E., Integration into the SDLC, The OWASP Foundation, https://www.owasp.org/images/f/f6/Integration_into_t he_SDLC.ppt
6. Kienzle, D. M and Elder, M. C., Final Technical Report: Security Patterns for Web

Application Development, http://www.scrypt.net/~celer/securitypatterns/final%20report.pdf, (2002)
7. Kenneth, R., Wyk, V., and McGraw, G. Bridging the Gap Software Development and Information Security, IEEE Security & Privacy, Vol. 3(5), pp. 75-79, (2005)
8. McGraw, G., Building Secure Software. A difficult but critical step in protecting your business, Citigal, Inc, http://www.cigital.com/whitepapers/dl/Building_Secu re_Software.pdf (2003)
9. McGraw, G., The Role of Architectural Risk in Software, Inform IT Network, http://www.informit.com/articles/article.aspx?p=446451
10. Microsoft Security Development Lifecycle, SDL Threat Modeling Tool, http://www.microsoft.com/security/sdl/adopt/threatm odeling.aspx
11. Mockel C and Abdallah, A.E, Threat Modeling Approaches and Tools for Securing Architectural Designs of E-Banking Application, Journal of Information Assurance and Security, Vol.6(5), pp346-356, (2010)
12. Mouratidis, H. and Giorgini, P., Security Attack Testing (SAT)- testing the security of information systems at design time, Information Systems, Vol. 32, pp1166- p1183, (2007)
13. OWASP Top 10, The Ten Most Critical Web Application Security Risk, http://owasptop10.googlecode.com/files/OWASP%20Top%2010%20-%202010.pdf
14. Pemmaraju, K., Lord, E. and McGraw, G. Software Risk Management. The importance of building quality and reliability into the full development lifecycle, Citigal, Inc., http://www.cigital.com/whitepapers/dl/wp-qandr.pdf (2000)
15. Ricard, R. (2011) ISO 1799 Risk Analysis Toolkit, http://sourceforge.net/projects/ratiso17799
16. Srinivasa, K.D. and Sattipalli, A. R, Hand Written Character Recognition using Back Propagation Network, Journal of Theoretical and Applied Information Technology, Vol. 5(3), pp257-269, (2009)
17. Swigart, S and Campell, S., Threat Modeling at Microsoft, http://download.microsoft.com/download/6/9/B/69BCB7C6-D158-4073-AD3E-F849E8ACBCE0/SDL_Series_-_4.pdf
18. Spampinato, D.G., SeaMonster: Providing Tool Support for Security Modeling, NISK Conference, http://www.shieldsproject.eu/files/docs/seamonster_ni sk2008.pdf
19. Ahmad, I., Swati, S.U. and Mohsin, S., Intrusion detection mechanism by resilient bpck Propagation (RPROP)", European Journal of Scientific Research, Vol. 17(4), pp523-530 (2007)
20. Liu, G., Hu, F. and Chen, W., A neural network emsemble based method for detecting computer virus,
In proceedings of 2010 International conference on computer, mechatronics, control and electronic engineering, Vol. 1, pp391-393 (2010)
21. Pan, Z, Chen, S., Hu, G. and Zhang, D., Hybrid neural network and c4.5 for misuse detection, In proceedings of 2003 International conference on machine learning and cybernetics, Vol.4, pp2463-2467, (2003)
22. Joseph, A., Bong, D.B.L. and Mat, D.A.A, Application of Neural Network in User Authentication for Smart Home Systems, World Academy of Science, Engineering and Technology, Vol. 53, pp1293- 1300. (2009)