2019 International Conference on Advancements in Computing (ICAC)
December 5-6, 2019. Malabe, Sri Lanka

Code Vulnerability Identification and Code Improvement using Advanced Machine Learning

Laneesha Ruggahakotuwa,Lakmal Rupasinghe,Pradeep Abeygunawardhana

Faculty of Computing, Sri Lanka Institute of Information Technology
Malabe, Sri Lanka
laneesha.or@gmail.com,lakmal.r@sliit.lk,pradeep.a@sliit.lk

***Abstract*—Cyber-attacks are fairly mundane. The misconfigurations of the source code can result in security vulnerabilities that potentially encourage the attackers to exploit them and compromise the system. This paper aims to discover various mechanisms of automating the detection and correction of vulnerabilities in source code. Usage of static and dynamic analysis, various machine learning, deep learning, and neural network techniques will enhance the automation of detecting and correcting processes. This paper systematically presents the various methods and research efforts of detecting vulnerabilities in the source code, starting with what is a software vulnerability and what kind of exploitation, existing vulnerability detection methods, correction methods and efforts of best researches in the world relevant to the research area. A plugin will be developed which is capable of intelligently and efficiently detecting the vulnerable source code segment and correcting the source code accurately in the development stage.**

***Index Terms*—Vulnerability, Machine learning, Deep learning,CVE**

I. Introduction

A software vulnerability can cause considerable damage to software s and businesses. Creating computer software is a complex and significant process that contains flaws and weaknesses. Verification of large codes may be ignored due to cost and its severity which results in unexpected, undesirable behaviors and system break downs. Despite the fact the severity of the software security is fickle, sometimes the recent high-profile exploitations such as Heart bleed bug, the Wanna Cry ransomware crypto worm, and the hack of the Equifax credit history database accomplish to exploit and cause critical damages such as drifting of productivity, intellectual property. These kinds of exploitations cause adverse effects on both finance and society. Although static and dynamic analysis tools are competent to find the limited subset of possible errors based on the predefined rules, the open-source repositories can discover code vulnerability patterns with the aid of data-driven techniques [1]. Identifying the vulnerabilities which affect the computer system is not limited to software. This research focuses on the software-based vulnerabilities instead of the hardware or system architecture and mainly contemplates the vulnerabilities of the library which used to develop the software. Although the automation of analyzing the software source code or binary plays a major role in finding the vulnerabilities in software vulnerability researches, currently most vulnerability discoveries are made via human interpretation. The security hole or weakness of the system design, implementation, operation and management that is accomplished to violate the systems security policy can be categorized as vulnerabilities in design or specification, vulnerabilities in implementation and vulnerabilities in operation and management [6]. When focusing on the vulnerabilities which are typical for many applications, hence adopt an attacker perspective, resisting the temptation of defining security policies to the targets [6]. In particular, the following policies can be considered. Code execution the client of the webserver should not allow executing the arbitrary code in the webserver with the kernel privileges. Information disclosure the raw memory of the remote program should not expose to unauthorized people. Denial of service. The attacker should not be permitted to terminate the execution of a system or a program on a remote host. The unauthorized should not be allowed to crash the kernel. Categorizing software vulnerabilities is not much simple task as it should be done during the analysis of the software error and codebase. Consistently the type can be anticipated during the discovery of vulnerabilities due to their techniques are implicitly targeted for a limited range of vulnerability types. Tools such as scanners detect the bad constructs and other security-related errors in the code. Most of the novel vulnerability prediction models are implemented based on machine learning techniques. Feature representing software codes are used as predictors for vulnerability [1]. Although multiple segments of codes have the same complexity metrics, they have different behaviors and the likelihood of the attack. The feature selection as the predictors should be done by the domain experts and should be prevented from outdated practices. Adopting manual interaction with the software vulnerability detection process is further decreased with the arrival of the machine learning techniques. Machine learning algorithms provide many solutions for previously potentially uncovered unknown vulnerabilities and it helps to design the syntax and semantics of the code and ascertain code patterns to analyze, collaborate in code auditing and understanding while achieving adequate false-positive rates in a manageable level. Using the static method to detect and analyze the vulnerabilities of source code is a crucial task since

978-1-7281-4170-1/19/$31.00 ©2019 IEEE

the programmers have their styles for developing applications.Since web app developers do not pay much attention to the security of front-end access it will affect the back-end database and scripts resulting in cross-site scripting, SQL injection and SQL manipulation. Machine learning is the approach used to analyzed software vulnerabilities. It is learned from and recognized from examples. This project proposes a software vulnerability detection and error correction mechanism using Deep machine learning algorithms. This will affect the efficiency and accuracy of the proposed project. Shallow methods will not give the intended accuracy levels required in the software development projects. Deep learning consisted of multi-layer operation with minimum time duration, maximum accuracy, and high performance. In here feature extraction is done in the training process. Extracting the noise from the data will be helpful for the implementation of an accurate and effective algorithm. The research will discuss multiple deep learning algorithms to enhance the detection of vulnerabilities and predicting the correct source code.

II. PROBLEM STATEMENT

Cyber-attacks are fairly mundane. In the worst case, the user is explained that the computer is encrypted and can be unlocked after the payment. According to the latest records, almost 60,000 data breaches have been notified. Most of the users consider the cause for this kind of threat is technology. The most severe data breaches such as Facebook, Uber, Equifax highlights that undiscovered code vulnerabilities can cause massive data breaches. It proves that the security of the software depends on its underlined source code. Intruders always hunt for the vulnerabilities of software. A small vulnerability such as misplacing a script tag, a single quote can cause extensive damage to its data. To overcome this problem Veracode, OWASP, Source Clear, etc. have become a solution in the industry. Always there is a limitation with cost, technology, and resources. Veracode caters both static and dynamic analysis, but it is highly expensive software. OWASP is limited to static analysis. Since most of the tools are in the testing phase, several false positives of the results can be very high and those results can be a lack of accuracy as well as the performance. Therefore, there is a need for an application that is capable of intelligently and efficiently detecting the vulnerable source code segment and correcting the sourcecode accurately in the development stage.

III. RESEARCH QUESTIONS

This research poses the following questions.

- How to automatically identify the software vulnerabilities of source code and the libraries by conducting a live scan, and detect error fragments?
- How to correct the detected vulnerabilities and fix the library of software at the code level?

IV. LITERATURE

This will explain the current contribution done by the researches to the detecting and analyzing the source code vulnerabilities. Jacob and others have introduced an automated software vulnerability detection applied for C and C++ mechanism using machine learning. They have proved that there is effectiveness in using machine learning at predicting the output of static analysis tools at the function level. By comparing the application of deep neural network models with more traditional models such as random forests, they founded that best performance is gained through the combination of features learned by deep models with tree-based models [17]. Hoa Khanh and others describe a new approach of automatic feature (semantic and syntactic) learning for vulnerability detection using a powerful deep learning LSTM model. Their evaluation on 18 android applications proved that the method of learning of features indicates 3 Fabian and others have proposed a method to automatic detection of unknown vulnerabilities of the source code and identify the API usage patterns using machine learning. They present two difficulties; first, how to catch the examples of API use naturally? Second, how to move these examples from known vulnerabilities to other code segments? [3]. They proposed a technique to render manual auditing increasingly viable by helping and managing the investigation of source code [3]. This extrapolation procedure comprises of four steps.1. Extraction of API symbols.2. Inserting in a vector space 3.Distinguishing proof of API use pattern.4. Helped vulnerability disclosure. [3]. Zhen Li et al has invented VulDeePecker as the first approach using deep learning for vulnerability detection. The main objective of this implementation was manually defining the features and reducing false negatives that acquired by other vulnerability detection systems. Most of the systematic experiments have proved that VulDeePecker can predict a very limited amount of false-negative rates than to the existing detecting systems [1]. Zhen Li and others have implemented a framework SySeVR using Deep learning to identify the vulnerabilities. It detected 15 vulnerabilities that were not identified in the NVD. Out of those 7 were unknown by the vendors and 8 were silently patched by the vendors. This is the high- level diagram of the proposed SySeVR framework.

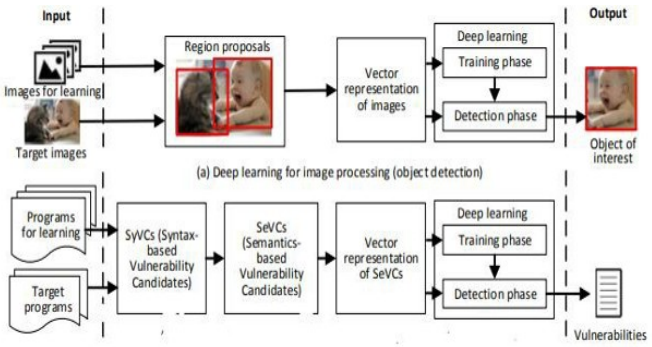

Fig. 1: SySeVR - A framework for using deep learning

Rebecca L. and others have utilized Deep representation Learning to automate vulnerability detection. [1].They have applied an assortment of machine learning systems for issue

grouping in the natural language domain, calibrating as outcomes utilizing a convolution neural network and characterized with an ensemble algorithm [1].

A. Vulnerability analyzing

Vulnerability analysis can be done in two ways such as statically and dynamically. The dynamic analyzers which over and again execute the projects with numerous test inputs and identify the weakness. [17]. The static analyzers identify the weakness of the source code without executing the program. These analyzers are rule-based tools with limited functionalities which unable to ensure the full test coverage of codes. Multiple approaches for lightweight scanning tools such as Flawfinder, RATS, ITS4, PScan and Microsoft PREfast have contributed towards the source code mining. These tools were able to discourage the use of problematic API s.

B. Detecting vulnerabilities

Detection of vulnerabilities can be done using static analysis tools based on machine learning, code matrix based on support vector machine for android devices, static and dynamic code aspects for web applications. Vulnerable components can be detected through N-gram analysis and feature selection methods [17]. The manual code review is an extravagant approach taken to vulnerability discovery. And Taint analysis is a good approach to detect the vulnerabilities and its false positives are identified by the data mining techniques. The researchers have leveraged the expert knowledge to reinforce the automated approaches to identify the vulnerabilities which are unique to each application. Evans, Larochelle leverage and Hackett et al have provided a glossary to discover vulnerabilities of the buffer overflow. In Java code Jif compiler allows us to check and confirm its security policies. This will interpret to reduce the appropriate amount of user-specified explanations. Researchers use query languages to identify irregularities. SourceClear, Blackduck, Veracode, OWASP Dependency Check, and Nexus Repository Pro are widely used vulnerability checking tools by the developers. Praxis uses contextual analysis in C, C++ and Java code to identify if function calls are correct and secure [5]. For example, strcpy() function in C and C++ which results in buffer overflows [5]. Once a risky function is identified it will keep a flag and identify whether the input variables would allow causing the attack [5]. The source code scanners such as app Sanctum, Kavados ScanDo, SPI Dynamics WebInspect, and Application Security AppDetective crawl websites and identify the presence of cross-site scripting, SQL injection and dictionary traversal attack. SPI Dynamics secure objects suite of Visual Studio .Net verify the user input by automatically blocking or forbid suspicious characters and known attacks used to bypass authentication forms.

C. Static Analysis Techniques

Static analysis is one of the techniques that detect the vulnerabilities in applications. This helps to identify the vulnerability before handing over to the targeted user and reduce the cost. This mechanism consists of three main steps such as analyze the input code, applying set rules or algorithms generate indicated by the application, and accomplishing the best a list of vulnerabilities available in the source code [14]. This method is very effective to detect the vulnerabilities of the buffer overflow. The static analysis method is used by many researches to make the vulnerability detecting process a success. Kremenk et al. has utilized machine learning to automatically surmise program details directly from programs by exploring the mix of source code program analysis and probabilistic graph models [7]. The ranking of possible errors according to their probabilities before examining any inferred specification can be done by creating an annotation factor graph using inference. Peng et al. has investigated the applying of deep learning for program analysis. [17]. And furthermore, they have done investigations by applying natural language granularity that can yield inadequate information and rather encode abstract syntax tree nodes into vector representations that classify a node in the AST as a solitary neural layer. At that point, they are utilized as the contribution for the Convolution Neural Network for deep supervised learning to classify programs [7]. K-means algorithm is used as a quantitative evaluation for clustering. [7]. The qualitative evaluation of classification based on deep learning shows a slight superior outcome compared to logistic regression and support vector machine. Yamaguchi et al. have done research on recognizing atypical or missing conditions connected to security-basic articles in the source code by statically taints source code [7].Discovery of neighborhood related functions are done by K-nearest neighbor algorithm [7]. Lightweight tainting pursued by the inserting of functions into vector space at that point permits correlation for missing checks by geometrically contrasting the check and the known checks for comparative capacities implanting in the rest of the codebase to identify the inconsistencies [7]. It was able to detect the known vulnerabilities in a higher detection rate and also able to detect the previously unknown vulnerabilities. Static code investigation is suitable for analyzing all the potential methods from the tainted source to conceivably vulnerable statements. It delivers false positives when working with complex language constructs or non-trivial sanitizing functions. It is delicate to the yields created by the compiler (static source code analyzer), yet it is genuine exploitability will be known simply after the compilation. [14].

D. Dynamic Analysis

Dynamic analysis is called a run time analysis. Although dynamic analysis produces very accurate results in vulnerability detection, it is very hard to find the relevant path to activate the vulnerability. VDiscover is a tool developed based on machine learning techniques used to extract the dynamic features from a binary to predict whether the vulnerability test case contains software vulnerability. Executing the traces which contain augmented functions with their parameters will result in the dynamic features. According to Tarmas et al. among the logistic regression, multi-layer perception, and random forest machine learning classification algorithms the random forest gives the best result in complex experiments.

E. Machine learning

When selecting a suitable machine learning algorithm for analyzing vulnerabilities, if the data is limited it is better to use semi-supervised learning. A prediction model has been built using CERT-C Secure Coding Standard. These are widely used approaches to mitigate source code level attacks. Maintain an inventory with all the libraries and protocols for the quick detection of the applications when something goes wrong. Fuzzing testing is very useful for companies that use immature and untested protocols in software and it is capable of identifying vulnerabilities in software, protocols, shared libraries and common file formats in the software. It takes inappropriate inputs and output an unexpected behavior and identify the error in the suspected program. Fuzzing is divided into dumb fuzzing and smart fuzzing [2]. Dump fuzzing is the simplest technique that uses randomly generated input values to the target software application [2]. This is a very effective method for test case creation since it can change the input values easily. Nonetheless, it is difficult to correctly identify the valid crash due to its narrow code coverage [2]. Smart fuzzing generates input values suitable for the format [2]. This can identify the location of the emerging of the errors through software analysis. Using fuzzing to analyze executable codes is the trend rather than using it for source code analysis. Other than this fuzzing can be used to fulfill various other tasks (Black box fuzzing, White box fuzzing, Gray box fuzzing). Harden the system which considers the compilation of software with control-flow integrity. Minimize the attack surface test is conducted to verify the security status of the product. When testing don't assume and make the decisions. As the above research summarizes, there are plenty of methods to detect the vulnerabilities of the source code. Among them, Machine learning is the most successful method. This proposed research uses deep learning-based machine learning for its success. As highlighted earlier, my approach to a novel architecture for robust analysis of source code by reducing the number of manual tasks and reduce the human interaction to detecting vulnerabilities of the source code. As the above research summarizes, there are plenty of methods to detect the vulnerabilities of the source code. Among them, Machine learning is the most successful method.

F. Bug fixing and Patching

Vulture is a tool that automatically learns from the areas of past vulnerabilities to foresee the future vulnerabilities of new segments before they are completely implemented. [7]. To detect and to match the patterns of the existing vulnerabilities Frequent pattern mining is used. To detect the unknown vulnerabilities Support vector machine method is used. Prophet is an automated patch generating framework that acquires patches from open-sourced software, repositories and creates the correct code. This procedure is found out in the offline training phase by highlighting effective patches extricated from past code amendment and it used to produce and prioritize the applicant patches for new imperfections [7]. Kazuki et al have built up a framework for source code appreciation strategies and metrics to foresee the understanding exertion in programming upkeep and advancement of the tasks. This study has consisted of two steps [18]. The first is to understand variant 1.0 of the application by perusing the clarification and source code by executing the application. This would be useful for the arrangement for future changes. The subsequent advance is to comprehend the change details, understand the comparing patches and choose whether the applied patches are right or not. This records the data, for example, time taken to see every alteration task whether the patch is right or not [18]. The motivation behind why they consider the applied fix is right or not and the perception procedure of the fix. They have done their investigation with programming improvement with a little paint application written in java with four semantics safeguarding alterations (refactoring, defect correction) and four semantics-changing adjustment tasks(enhanced and change) [18].

Qosai et al have done an examination on the effect of improving the source code on programming measurements. They have assessed the capability to break down the procedures and tools of programming source code, identifying the potential imperfections of the product items. And furthermore, they have assessed the effect of applying alterations suggested by programming code analyzing tools (SCA) on programming measurements [19]. The selected metrics center around the number of lines in the source code, for example, Lines of code (LOC), Comments, BLOC. Because of looking at Just- Code and StyleCop SCA instruments shows that the StyleCop issues are more with the issues identified with the refactoring while JustCode assessment issues are more profound towards the structure, quality, and viability issues [19].

V. PROPOSED METHODOLOGY

As shown in Fig.2 the proposed methodology consists with two phases.

- Error Detection
- Error Correction

As shown in Fig.3 Error Detection is done through a live scan. In live scan both Static analysis and Dynamic Analysis will run parallelly.

In the dynamic analysis, the source code will run in the background and will check with random input data. If it is vulnerable for specific attacks related to the input randomdata, It will show an Error message. If it shows the error message it will be verified with the rule-based engine with the input Random data and identify the vulnerable code segment.

In static analysis, the source code will be checked line by line and verified with the Rule-Based Engine. The source code will be highlighted with warning signs based on the two outputs of static and dynamic analysis. To cater to the Error

prevention method, the most suitable machine learning algorithm should select from Nave Bayes, Bayesian networks, k-NN, HMM, ANN, SVM, and Deep learning.

After the most suitable model is identified, the model will be trained with enough training samples to develop a generalized model. Validation and test data sets will be used to evaluate the prediction models. The models will be evaluated on the validation and test data. The overall system performance will be evaluated. The final system will be implemented for Java initially.

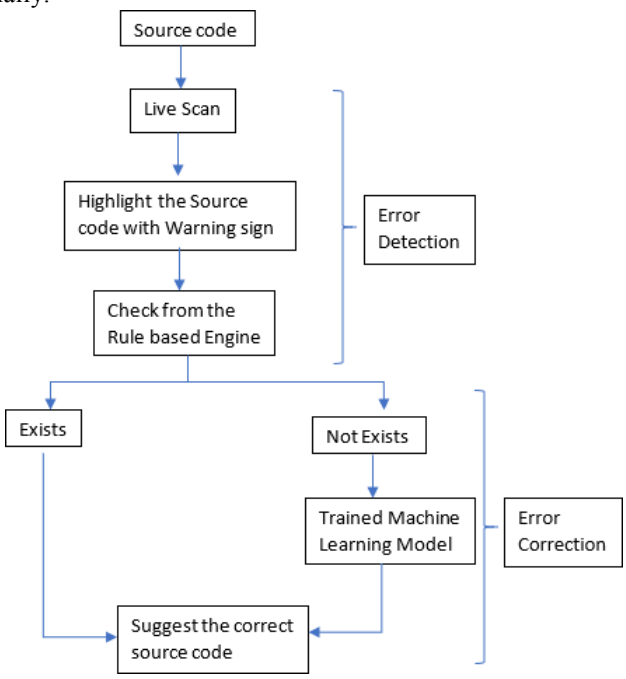

Fig. 2: High Level System Diagram

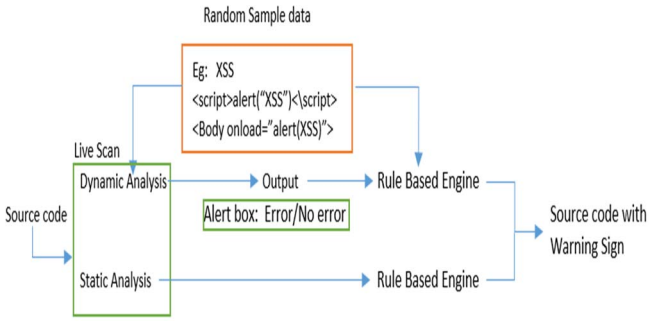

Fig. 3: Error Detection

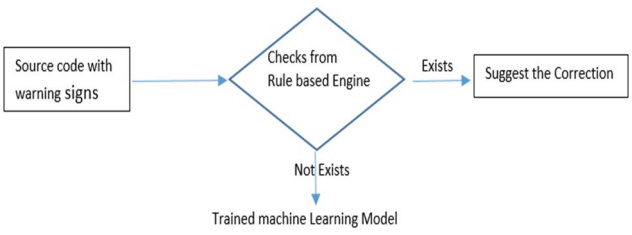

Fig. 4: Error Correction

VI. RESULTS AND DISCUSSION

This section focuses on the results of this project committed to detect the OWASP top 10 vulnerabilities. Two of them are briefly described as a sample. Following is a sample source code which is written by the developer. When the developer writes the code, the system detects the vulnerable code lines and propose the corrected source code segment.

```
<?php
require_once('../header.php');
require_once('db.php');
$sql ="SELECTFROMuserswherename='";
$sql = $_GET["name"]."";
$result=mysql_query($sql);
if($result)
{
?>
```

A. SQL Injection Attack

The attacker can use ' or '1'='1 payload to make the exploit. This makes $sql SELECT * FROM users WHERE name = or 1 = 1, which returns the dump of data of a specific table. When the developer clicks on that warning sign, the system will introduce the corrected source code segment as below.

```
<?php
if(preg matah('//',$GET["name"])){
die("ERRORNOSPACE");
}
$sql ="SELECT  * FROMuserswherename='";
$sql = $_GET["name"]."'";
$result = mysql_query($sql);
?>
```

B. Cross site Scripting

The attacker can inject a malicious script to display an alert box. <script>alert("you are hacked")< script>. When the developer clicks on the warning sign the system will suggest the corrected source code as below by filtering the script tag.

```
<?php
$name = $_GET["name"];
$name = preg_replace("/<script>/","",$name);
$name = preg_replace("/<\/script>/","",$name);
echo $name;
?>
```

This section explains the research findings achieved during the error detecting phase. These are the evidence that proves

how this research has achieved its primary objectives with constraints such as time, knowledge, standards. The following are the results Fig.5, Fig.6, Fig.7 the test cases of training, testing and predicting phases of the above system. These are well planned, efficient and accurate searching methods.

The test case 01 which shows the quantitative results comparing the estimated and execution time for neural network training.

Test case 01 : command execution for Neural Network Training				
	Step actions	Expected results	Actual results	status
1	Execution of command "sudo python app/main.py --training"	Terminal Message "training start"	Terminal Message "training start"	passed
2	Execution command if incorrect	Should display relevant error message	Displayed an error message	passed
Execution type	Manual			
Estimated exec. duration (min)	Less than 1.00			
Last Result	Passed			
execution duration (min)	1.00			
Build	New build			
Tester	Developer			
	Actual Results – passed training start code property graph is creating... Warning:your JVM has a maximum heap size of less than 2 Gig. You may need to import large code bases in batche If you have additional memory, you may want to allow your JVM to access it byusingthe -Xmx flag.			

The test case 02 which shows the quantitative results comparing the estimated and real time, results for neural network testing.

Fig. 5: Test case - Training results

Test case 02 : Command execution for Neural Network Testing				
	Step actions	Expected results	Actual results	status
1	Execution of command "sudo python app/main.py --testing"	Terminal Message "testing start"	Terminal Message "testing start"	passed
2	Execution command if incorrect	Should display relevant error message	Displayed an error message	passed
Execution type	Manual			
Estimated exec. duration (min)	1.00			
Last Result	Passed			
execution duration (sec)	53.00			
Build	New build			
Tester	Developer			
	Actual Results – passed training start code property graph is creating... Warning:your JVM has a maximum heap size of less than 2 Gig. You may need to import large code bases in batches If you have additional memory, you may want to allow your JVM to access it byusingthe -Xmx flag.			

Fig. 6: Test case - Testing results

Test case 03 : Command execution for Neural Network Prediction				
	Step actions	Expected Results	Actual Results	status
1	Execution of command "sudo python app/main.py --predict"	Terminal Message "predict start"	Terminal Message "predict start"	passed
2	Execution command if incorrect	Should display relevant error message	Displayed an error message	passed
Execution type	Manual			
Estimated exec. duration (min)	1.00			
Last Result	Passed			
execution duration (sec)	50.00			
Build	New build			
Tester	Developer			
	Actual Results – passed training start code property graph is creating... Warning:your JVM has a maximum heap size of less than 2 Gig. You may need to import large code bases in batches If you have additional memory, you may want to allow your JVM to access it byusingthe -Xmx flag.			

Fig. 7: Test case - Predicting results

The test case 03 which shows the quantitative results comparing the estimated and real time, results for neural network prediction.

VII. CONCLUSION

The solution will be developed as an open-source plugin for Net beans. It can detect the vulnerable source code segments accurately and efficiently with a warning sign. The identified vulnerable source code is replaced with the correct source code.

REFERENCES

[1] H. Rathore, S. Agarwal, S. K. Sahay, and M. Sewak, "Malware Detection using Machine Learning and Deep Learning.," *Cryptography and Security (cs.CR); Machine Learning (cs.LG)*, vol. 11297, pp. 402-411, 2018.

[2] H. K. Dam, T. Tran, T. T. M. Pham, S. W. Ng, J. Grundy, and A. Ghose, "Automatic feature learning for predicting vulnerable software components," IEEE Transactions on Software Engineering, Institute of Electrical and Electronics Engineers Inc., 2018.

[3] J. Jurn, T. Kim, and H. Kim, "An automated vulnerability detection and remediation method for software security",Sustain., vol. 10, no.5, May 2018.

[4] N. Shakhovska, "Advances in Intelligent Systems and Computing", Ukraine: Springer, Cham, 2016.

[5] I. P. L. Png, C.-Y. Wang, and Q.-H. Wang, "The Deterrent and Displacement Effects of Information Security Enforcement: International Evidence", *Journal of Management Information Systems* , vol. 25,Taylor Francis, Ltd., pp. 125144.

[6] H. Nguyen, H. Dang, T. Le, and S. Le, "Innovative Mobile and In- ternet Services in Ubiquitous Computing - Proceedings of the 11th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS-2017)", Torino, Italy, 10-12 July 2017, vol. 612, 2018.

[7] C. M. Chase and S. A. Jacob Abraham,Ernest A Emerson II Vijay K Garg Aleta M Ricciardi "Testing Concurrent Software Systems Commit- tee."

[8] H. Liang, X. Pei, X. Jia, W. Shen, and J. Zhang, "Fuzzing: State of the Art," IEEE Trans. Reliab., vol. 67, no. 3, pp. 11991218, Sep. 2018.

[9] HT. B. Lee, *Which Languages Are Bug Prone* tim@vox.com, 2014.[Online]. Available: https://www.i-programmer.info/news/98-languages/11184-which-languages-are-bug-prone.html

[10] T. Abraham and O. De Vel, "A Review of Machine Learning in Software Vulnerability Research,",Defence Science and Technology Group, Australia, 2017.

[11] R. Amankwah, P. K. Kudjo, and S. Y. Antwi, "Evaluation of Software Vulnerability Detection Methods and Tools: A Review," Int. J. Comput. Appl., vol. 169, no. 8, pp. 2227, 2017.

[12] M. Dowd, J. McDonald, and J. Schuh, "An automated vulnerability detection and remediation method for software security", Pearson Education, 2006.

[13] F. Yamaguchi, "Pattern-Based Vulnerability Discovery", University of Gttingen, 2015.

[14] H. K. Dam, T. Tran, T. Pham, S. W. Ng, J. Grundy, and A. Ghose, "Automatic feature learning for vulnerability prediction",arXiv Prepr.arXiv1708.02368, 2017.

[15] T. B. Lee, "The Heartbleed Bug", explained - Vox, tim@vox.com,2014.[Online].Available:https://www.vox.com/2014/4/8 /5593654/ heartbleedexplainer-big-new-web-security-flaw-compromise-privacy

[16] C. Fenton, "How to Check Open Source Code for Vulnerabilities DZone Security" Security Zone Analysis, 2017. [Online].Available:https://dzone.com/articles/how-to-check-open-source-codefor-vulnerabilities.

[17] R. Russell, "Automated vulnerability detection in source code using deep representation learning," in 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), 2018, pp.757762.

[18] K. Nishizono, S. Morisakl, R. Vivanco, and K. Matsumoto, "Source code comprehension strategies and metrics to predict comprehension effort in software maintenance and evolution tasks," *an empirical study with industry practitioners*, in 2011 27th IEEE International Conference on Software Maintenance (ICSM), 2011, pp.473481.

[19] Q. Zoubi, I. Alsmadi, and B. Abul-Huda, "Study the impact of improving source code on software metrics", in 2012 International Conference on Computer, Information and Telecommunication Systems (CITS), 2012,